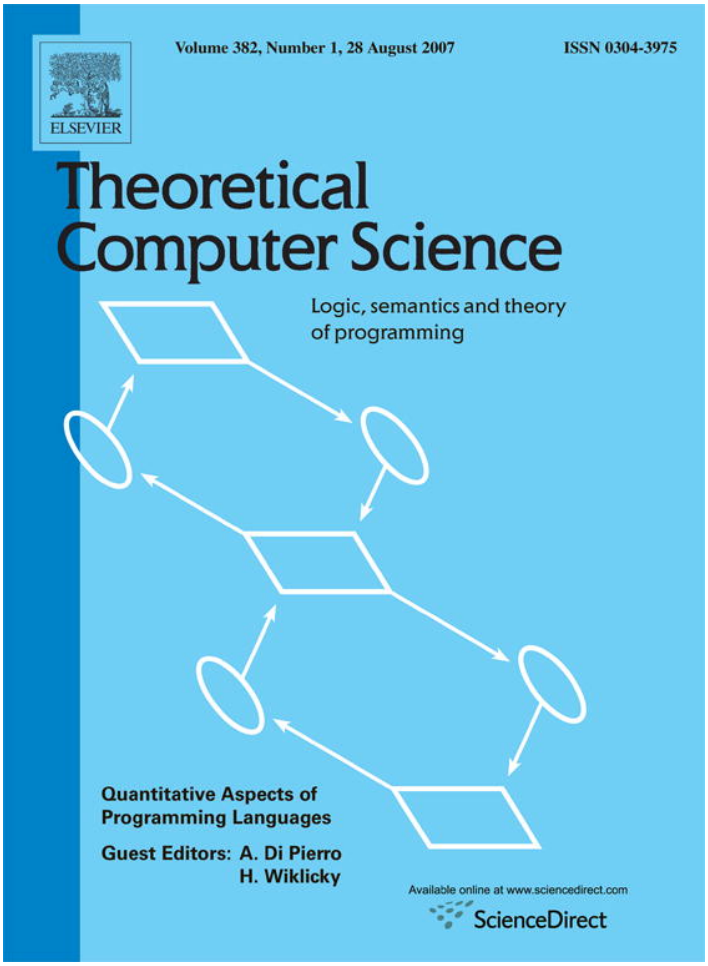

Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 382 (2007) 42–70
Theoretical Computer Science
www.elsevier.com/locate/tcs

Model checking mobile stochastic logic☆

Rocco De Nicola[a], Joost-Pieter Katoen[b], Diego Latella[c], Michele Loreti[a,*], Mieke Massink[c]

[a] *Università degli Studi di Firenze, Italy*
[b] *RWTH Aachen University, Germany*
[c] *C.N.R. - I.S.T.I, Pisa, Italy*

Abstract

The Temporal *Mo*bile *S*tochastic *L*ogic (MoSL) has been introduced in previous work by the authors for formulating properties of systems specified in StoKlaim, a Markovian extension of Klaim. The main purpose of MoSL is to address key functional aspects of global computing such as distribution awareness, mobility, and security and their integration with performance and dependability guarantees. In this paper, we present MoSL$^+$, an extension of MoSL, which incorporates some basic features of the Modal Logic for MObility (MoMo), a logic specifically designed for dealing with resource management and mobility aspects of concurrent behaviours. We also show how MoSL$^+$ formulae can be model-checked against StoKlaim specifications. For this purpose, we show how existing state-based stochastic model-checkers, like e.g. the Markov Reward Model Checker (MRMC), can be exploited by using a front-end for StoKlaim that performs appropriate pre-processing of MoSL$^+$ formulae. The proposed approach is illustrated by modelling and verifying a sample system.

Keywords: Stochastic process algebra; Mobility; Global computing; Stochastic logics; Stochastic model-checking

1. Introduction

1.1. Global computing

During the last couple of decades, computer systems have changed significantly: stand-alone, static devices executing programs autonomously have evolved into large-scale networks of computing devices performing tasks in a cooperative and coordinated manner. These modern, complex distributed systems – also known as *global or network-aware computers* [12] – are highly dynamic and have to deal with frequent changes in the network environment.

☆ The work presented in this paper has been partially supported by the EU Project Software Engineering for Service-Oriented Overlay Computers (SENSORIA, contract IST-3-016004-IP-09) and by FIRB-MUR project Tecnologie orientate alla conoscenza per aggregazioni di imprese in internet (TOCAI.IT).

* Corresponding address: Universita degli Studi di Firenze, Dipartimento di Sistemi ed Informatica, Viale Morgagni, 65, 50134 Firenze, Italy. Tel.: +39 055 4237449; fax: +39 055 4237436.
E-mail address: loreti@dsi.unifi.it (M. Loreti).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.05.008

The world wide web is a major example of a global "computer". Features such as distribution awareness and code mobility, which were absent or deliberately invisible in previous computer generations, play a prominent role in global computing. Dedicated programming and specification formalisms have been developed that can deal with issues such as (code and agent) mobility, remote execution, security, privacy and integrity. Important examples of such languages and frameworks are, among others, Obliq [11], Seal [14], ULM [9] and KLAIM (*Kernel Language for Agents Interaction and Mobility*) [16,6].

1.2. Dependable global computing

Performance and dependability issues are of the utmost importance for "network-aware" computing, due to the enormous size of systems – networks typically consist of thousands or even millions of nodes – and their strong dependence on mobility and interaction. Spontaneous computer crashes may easily lead to a failure of remote execution or process movement, while spurious network failures may cause the loss of code fragments or unpredictable delays. The enormous magnitude of computing devices involved in global computing yield failure rates that can no longer be ignored. The presence of such random phenomena implies that the correctness of global computing software and their safety guarantees are no longer rigid notions like:

> "either it is safe or it is not"

but have a less absolute nature, e.g.:

> "in 99.7% of the cases, safety can be ensured".

The intrinsic complexity of global computers, though, complicates the assessment of these issues severely. Systematic methods, techniques and tools—all based on solid mathematical foundations i.e., *formal methods*, are therefore needed to establish performance and dependability requirements and guarantees. This paper attempts to make a considerable step in this direction by proposing an extension of a widely used temporal logic, CTL, as a property specification language for distribution, performance and dependability guarantees. The temporal logic formalism presented in the present article builds upon an action-based variant of CSL (Continuous Stochastic Logic [2,5]).

1.3. Modelling dependable global computing

To facilitate the incorporation of random phenomena in models for network-aware computing, we proposed STOKLAIM [17], a simple, yet powerful stochastic extension of KLAIM [16,6], an experimental language for distributed systems that is aimed at modelling and programming mobile code applications, i.e., applications for which exploitation of code mobility is the prime distinctive feature.

In STOKLAIM, every action has a random duration governed by a negative exponential distribution. The resulting operational model is therefore a continuous-time Markov chain (CTMC, for short), one of the most popular models for the evaluation of the performance and dependability of information processing systems. Our extension is inspired by Markovian extensions of traditional process algebras; for recent surveys see, e.g. [28,32]. A preliminary version of the language STOKLAIM has been published in [20].

1.4. Specifying properties of dependable global computing

To assess dependability aspects, typically long-run or transient probabilities of CTMCs are considered; we propose to adopt a more recent technique that determines performance and dependability *guarantees* in a fully automated manner using model checking. Guarantees are formulated as temporal logic formulae. For CTMCs, the logic CSL (Continuous Stochastic Logic) [2,5] is of particular interest, and efficient model-checking algorithms exist for it. CSL is a stochastic extension of CTL that, together with qualitative properties, permits specifying time-bounded probabilistic reachability properties, such as "the likelihood of reaching a goal state within t time units while visiting only legal states is at least 0.92". Several software tools have been developed for supporting the verification of CSL formulae; here, we just mention PRISM [35], ETMCC [31] and MRMC [33].

A clear advantage of the logical approach to performance and dependability assessment is the completely *formal* characterisation of the performance and dependability measures of interest. Informal descriptions of complex

measures could in fact be easily misinterpreted and more error prone. Moreover, with the help of a stochastic model-checker, one can automatically check whether a performance or dependability requirement is fulfilled by a specific system model. It is important to point out that model-checking tools not only provide a *yes/no* answer, but provide *also* the values of the probabilities of interest. In this sense, stochastic model-checkers incorporate also the functionality of traditional Markov Chain analysis tools, but such a functionality is embedded in a general formal framework. Also, functional properties of behaviour, usually expressed by temporal logics like CTL, can be often characterised by formulae of *stochastic* temporal logics, where the degenerate probability values 0 and 1 are used. This means that stochastic logics permit formulating and automatically checking *both* functional *and* non-functional properties of system behaviour in an *integrated* way, *with the same* formalism.

In [18], we proposed MoSL (*Mobile Stochastic Logic*), a logic that allows one to refer to the spatial structure of the network for the specification of properties of StoKlaim models. A preliminary version of the logic was presented in [19]. Our starting-point was an action-based variant of CSL (as first proposed in [30]), that fits well with the action-based nature of Klaim. The distinguishing features of MoSL, with respect to CSL, are:

- atomic propositions may refer to the sites where data and processes reside,
- actions are generalised to *action specifiers* that act as patterns for characterising sets of actions, and
- logical variables are incorporated to refer to dynamically created sites.

In this paper, we present MoSL$^+$, an extension of MoSL, which incorporates some basic features of the *Mo*dal Logic for *Mo*bility, MoMo [22], that has operators for describing properties resulting from resource production and consumption. In particular, in the new logic, state properties incorporate features for resource management and context verification, namely the MoMo *consumption* and *production* operators. Context verification allows the verification of assumptions on resources and processes in a system at the logical level, i.e. without having to change the model to investigate the effect of each assumption on the system's behaviour. We also show how, by an appropriate use of the techniques presented in [30], any formula of MoSL$^+$, including those with binding occurrences of variables, can be model-checked using the state-based model-checker MRMC. For the actual model checking, we developed a prototype front-end tool, named SAM (Stochastic Analyser for Mobility).

1.5. Related work

Probabilistic CTL (PCTL, for short) is a branching-time temporal logic for discrete-time Markov processes [26, 8]. It has been applied to case studies from different fields ranging from distributed systems to systems biology and security, and various variations and extensions to PCTL have been defined [3,15]. CSL is the counterpart of PCTL for continuous-time Markov chains, and originates from [2,4] and basically inherits the probability operators from PCTL, extends these with a steady-state operator, and uses real-time variants of until (as they occur in timed CTL [1]).

Several (temporal) logics have been proposed which aim at describing properties of systems related either to mobility ([7,21,10,13,25,36] among others) or to probabilistic/stochastic behaviour (e.g. [26,27,2,5,30]). To the best of our knowledge, [18,19] is the first approach towards a probabilistic logic for mobility which is closely related to the language presented in [20].

In [4], an action- and state-based stochastic logic is proposed. The specification of path properties makes use of regular expression operators which cover, among other things, the functionality of until and next. The logic is based on uninterpreted states and actions. A model-checking procedure is proposed which uses CSL model-checking. Neither distribution awareness and mobility, nor network resource management, are addressed in the above mentioned paper.

1.6. Structure of the paper

Section 2 briefly recalls the modelling language StoKlaim, the use of which is shown by means of a simple example which will be used throughout the paper. The property specification language MoSL$^+$ is introduced in Section 3 together with its formal semantics. An algorithm for using the MRMC model-checker for the logic CSL to model-check MoSL$^+$ formulae against models specified in StoKlaim is presented in Section 4. Section 5 shows how several interesting properties of the example model can be automatically verified. Finally, in Section 6 some conclusions are drawn and lines of future research are outlined.

Table 1
Syntax of StoKlaim nets

N	::=	$\mathbf{0}$	|	$i ::_{\rho} E$	|	$N \parallel N$				
E	::=	P	|	$\langle \vec{f} \rangle$						
P	::=	$\mathbf{nil}$	|	$(A, r).P$	|	$P + P$	|	$P \mid P$	|	$Q(\vec{P}, \vec{\ell}, \vec{e})$
f	::=	P	|	ℓ	|	e				
A	::=	$\mathbf{newloc}(!u)$	|	$\mathbf{out}(\vec{f})@\ell$	|	$\mathbf{in}(\vec{F})@\ell$	|	$\mathbf{read}(\vec{F})@\ell$	|	$\mathbf{eval}(P)@\ell$
F	::=	f	|	$!X$	|	$!u$	|	$!x$		

2. StoKlaim

This section briefly recalls the language StoKlaim and introduces the basic notation used throughout the paper. The full details of the formal definition of the language and a thorough discussion of the motivations of all our design choices can be found in [17].

We provide a gentle introduction to StoKlaim by presenting the key constructs of the language by means of a small but representative example of a generic Distributed Mobile Service (DMS), which we describe below, and which we will use as a running example throughout the paper.

A DMS is a network service that exploits the capabilities of different network resources. We present a limited configuration of the service for the purposes of concise presentation. The service relies on two sites, say A and B. Client software is assumed to run only on A. On site A, a service dispatcher is also running that receives service requests from local users and dispatches them to the appropriate sites. There are two types of services, $S1$ and $S2$. $S1$-type service is a simple service that requires only local resources. $S2$-type service requires first resources in A, and then resources in B. An example of an $S2$-type service could be a number-crunching application consisting of two phases: a preparatory one, completely performed locally, followed by a second phase which requires specialised computing resources, not available locally.

We now move to the informal description of StoKlaim. Like in Klaim, a StoKlaim *network* consists of a set of *sites*, each having a *physical address*, a set of running *processes*, and a collection of *stored tuples*, which can be basic data *values* or processes. Consequently, we need a set $\mathcal{I}$ of (physical) *addresses*, ranged over by $i, i', i_1, \ldots$, and a set $\mathcal{P}$-var of *process variables*, ranged over by $X, X', X_1, \ldots, Q, Q', Q_1, \ldots$. We will conventionally use $Q, Q', Q_1, \ldots$ for those process variables for which there is a definition in the StoKlaim specification at hand, as described in a moment. Moreover we assume a set $\mathcal{V}$ of (basic data) *values*, ranged over by $v, v', v_1, \ldots$, a set $\mathcal{V}$-var of *value variables*, ranged over by $x, x', x_1, \ldots$, and a standard way for building value expressions from values, value variables and operators; in the following, we let e denote any generic value expression, and we do not discuss these expressions in any further detail here.

Processes can *upload/read/download* tuples to/from/from sites as well as execute network management actions like *creating* new sites, and *spawning* other processes to (remote) sites. Tuples are retrieved from tuple spaces via *pattern matching* using *templates*. Templates are sequences of *actual* and *formal* fields, or *binders*, which are *variables* that will get a value when a tuple is retrieved. Formal fields are marked by a '!' before the variable name. In order to guarantee a high level of flexibility and portability of the process code, processes, in their actions, cannot use directly (physical) addresses, but they can refer to sites only by means of *logical addresses*, usually called *localities*. We let $\mathcal{L}$, ranged over by $l, l', l_1, \ldots$ be a set of *localities*, and $\mathcal{L}$-var, ranged over by $u, u', u_1, \ldots$ be a set of *locality variables*. The association of localities to addresses, is *local* to sites; each site is equipped with an *allocation environment* which is a mapping used for resolving the localities used by the processes running at that site, translating them to addresses. In StoKlaim, process actions are enriched with *rate-names* which characterise the *duration* of the actions. More specifically, the rate-name occurring in an action specifies, via a global *rate-mapping*, the rate of an exponentially distributed random variable characterising the duration of the execution of the action. The use of rate-names and rate-mappings instead of direct rates, i.e. real numbers, facilitates the systematic analysis of a given network under several, different, timing assumptions, since this requires only modifications to the rate-mapping. Moreover, there are also technical reasons for preferring rate-names to rates. The reader interested in further details on this issue is referred to [17]. We let $\mathcal{R}$, ranged over by $r, r', r_1, \ldots$, be the set of *rate-names*. All the above sets are assumed countable and mutually disjoint. Furthermore, let ℓ, ℓ', ℓ_1 range over $\mathcal{L} \cup \mathcal{L}$-var.

In the syntactical definition of StoKlaim main constructs, recalled in Table 1, we adopt the $\vec{(\cdot)}$-notation for sequences; e.g., $\vec{l} = l_1, l_2, \ldots, l_n$ denotes a sequence over $\mathcal{L}$ and $\vec{x} = x_1, x_2, \ldots, x_m$ is a sequence over $\mathcal{V}$-var. For sequence $\vec{s} = s_1, \ldots, s_n$, let $\{\vec{s}\}$ denote the set of elements in $\vec{s}$, i.e., $\{\vec{s}\} = \{s_1, \ldots, s_n\}$. One-element sequences and singleton sets are denoted as the elements they contain, i.e., $\{s\}$ is denoted as s and $\vec{s} = s'$ as s'. The empty sequence is denoted by ϵ. In this paper, we will often use a functional programming-like notation, where currying will be used in function application, i.e., for function foo, foo $a_1\ a_2 \ldots\ a_n$ will be used instead of $\mathsf{foo}(a_1, a_2, \ldots, a_n)$, and function applications will be considered left-associative. We let (dom foo) denote the *domain* of foo.

2.1. Nets and processes

A network state is modelled in StoKlaim by means of a *net* expression N (see Table 1). The most elementary net is the null net, denoted $\mathbf{0}$. A net consisting of a single node with *address* i is denoted $i ::_\rho E$, where ρ is an *allocation environment* and E is a *node element*. The allocation environment ρ is a partial function from $\mathcal{L}$ to $\mathcal{I}$ mapping the localities occurring in the processes running at node i to addresses. Notice that the operational semantics of StoKlaim postulates that whenever a process uses a locality on which the allocation environment is undefined, the process deadlocks. Nets may be composed of the parallel composition of several nodes. Node elements are either processes executing at the node – *process nodes* in the sequel – or data (represented as a tuple $\vec{f}$) that is stored at the node.

So, a network state is modelled in StoKlaim as a *net* N. Notice that, in general, in N there can be more than one node with the same address i. The *site* (with address) i in the network is modelled by the collection of nodes in N with address i. Nodes are syntactic objects, whereas sites are conceptual entities. The set of processes *running at* site i is the set of processes P such that $i ::_\rho P'$ occurs in N and $P = P'$, or is P a proper sub-process of P'. The set of processes (localities, or basic values, respectively) *stored at* site i is the set of processes (localities, or basic values respectively) occurring as fields of tuples $\vec{f}$ such that $i ::_\rho \langle \vec{f} \rangle$ is in N.

We can now give a first, abstract, definition of our DMS. We assume the system is originally created at an existing site, with a conventional address $init \in \mathcal{I}$, and we let the initialisation phases be performed by the process *Boot*. The initial state of our system is described by the following net:

$$init ::_{[\mathsf{self} \to init]} Boot \tag{1}$$

where [self $\to$ *init*] denotes the allocation environment which maps self $\in \mathcal{L}$ to *init*. In general, we assume the existence of self $\in$ (dom ρ) for all allocation environments ρ, and require ρ self $= i$ for node $i ::_\rho E$. Before further specifying the process *Boot*, we briefly describe the syntax and informal semantics of StoKlaim *processes* P.

Processes are built up from the terminated process **nil**, a set of randomly delayed actions, and standard process algebraic operators such as prefix, choice, parallel composition and process instantiation Q, with optional parameters $(\vec{P}, \vec{\ell}, \vec{e})$, where the process variable Q is assumed to be defined in the sequence of process definitions $\vec{D}$ in the StoKlaim specification at hand, by a process defining equation of the form:

$$Q(!\vec{X}, !\vec{u}, !\vec{x}) \stackrel{\Delta}{=} P.$$

For syntactical clarity, *all* binding occurrences of variables are prefixed with '!'. This includes occurrences as arguments of node creation operation, of in/read actions, and as formal parameters of process definitions. In this paper, we require that each process instantiation be *action guarded*, i.e. prefixed by an action.

The process $(A, r).P$ executes action A with a duration that is a random variable which is exponentially distributed, with a rate specified by rate-name r. Rate-names are mapped to rate values by means of *rate-mappings*, to generate an action-labelled CTMC, to be used for formal analysis and verification. A rate-mapping β is a partial function from $\mathcal{R}$ to $\mathbb{R}_{>0}$; thus, the duration of the execution of action A is a random variable with a negative exponential distribution, with rate $(\beta\, r)$.

It is worth pointing out here that, although fairly simple, the mechanism of rate-mappings is quite powerful and flexible. In fact, we can extend the domain of rate-mappings in such a way that it includes all the information characterising the actions processes execute, like the addresses of the sites where the actions are executed, those of the target sites (e.g. the site where a tuple is uploaded or a process is spawned), the arguments of the specific actions etc. This way, one can make the (parameters of the random) durations of the actions depend, for instance, on the rate-name

used in the specific action, and/or the site where they are executed, and/or the size of involved data, and/or the target site etc. In this paper, for the sake of simplicity, we let rate-mappings depend only on the rate-names.

Notice also that we did not include a *probabilistic* choice operator; in this paper, we follow the traditional approach of Markovian process algebras, where the probabilities of alternative branches of behaviour are *derived* from action rates on the basis of the race condition principle. In other words, all instances of non-determinism which may arise from the choice and the parallel operators are resolved on the basis of the speed of the process actions involved: the action with the shortest duration is chosen. We leave the extension of StoKlaim with probabilistic choice and parallel composition as well as the study of the interactions of these constructs with rates for further study.

2.2. Actions

A process can create a new site by means of the action **newloc**$(!u)$. This action will have also the effect of creating a fresh new address, say i, and a fresh new locality, say l. The newly created locality l will be bound, in the allocation environment ρ of the node where the action is executed, to the address i, and all the free occurrences of u are replaced with l. The resulting allocation environment will be exported to the newly created node.

In order to see how the **newloc** operates, let us go back to our running example. Suppose process *Boot* starts by creating the two sites of the DMS, being defined as follows, where process *Boot*1 will be further specified later:

$$Boot \stackrel{\Delta}{=} (\mathbf{newloc}(!x), d1).(\mathbf{newloc}(!y), d2).Boot1(x, y).$$

Assuming that the fresh address "A" and locality "a" are generated by the first action of *Boot*, the system evolves from the initial state above to the following net[1]:

$$init ::_{\rho_1} (\mathbf{newloc}(!y), d2).Boot1(a, y) \parallel A ::_{\rho_2} \mathbf{nil} \tag{2}$$

where

$$\rho_1\, l \stackrel{\text{def}}{=} \begin{cases} init, & \text{if } l = \mathsf{self} \\ A, & \text{if } l = a \\ \text{undefined}, & \text{otherwise} \end{cases} \qquad \rho_2\, l \stackrel{\text{def}}{=} \begin{cases} A, & \text{if } l \in \{\mathsf{self}, a\} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

As usual, system evolution can be formalised by means of a *labelled transition system* (LTS), whose states are basically StoKlaim nets. In practice, for technical reasons, *configurations* are used instead of nets: a configuration is a net enriched with additional information concerning the addresses and localities used in the current state. For the sake of notational simplicity, in the sequel we disregard such additional information, unless strictly necessary. The transition over states is labelled with information concerning the specific operation which caused an evolutionary step. More specifically, $N_1 \xrightarrow{\gamma, r} N_2$ should be read as: the system may evolve from configuration N_1 to configuration N_2 by means of the execution of the action described by γ, whose duration is determined by r. It is worth pointing out here that the transition relation represents action execution only *symbolically*, due to the presence of the rate-names, instead of rates. Real, concrete, execution requires that samples are drawn of the random variables, exponentially distributed with rates determined by a rate-mapping applied to the rate-names occurring in the transitions. We will come back to these notions in Section 3, when defining *paths* over the Markov Chain obtained from the LTS associated to a StoKlaim net and to a rate-mapping.

The initial step of our running example is formalised by the following element of the transition relation:

$$init ::_{[\mathsf{self} \to init]} Boot \xrightarrow{(init, \mathbf{n}(A)), d1} init ::_{\rho_1} (\mathbf{newloc}(!y), d2).Boot1(a, y) \parallel A ::_{\rho_2} \mathbf{nil}$$

where the information γ on the action is $(init, \mathbf{n}(A))$, describing the fact that the action of concern is a **newloc**, which is executed at site *init* and generates a site with address A. Assuming now that the fresh address "B" and locality "b" are generated by the next action of *Boot*, which is again a **newloc**, the behaviour of the system will continue with the following transition:

$$init ::_{\rho_1} (\mathbf{newloc}(!y), d2).Boot1(a, y) \parallel A ::_{\rho_2} \mathbf{nil} \xrightarrow{(init, \mathbf{n}(B)), d2}$$
$$init ::_{\rho_3} Boot1(a, b) \parallel A ::_{\rho_2} \mathbf{nil} \parallel B ::_{\rho_4} \mathbf{nil}$$

[1] The specific way in which fresh names are generated is of no interest in the context of the present paper.

where

$$\rho_3\, l \stackrel{\text{def}}{=} \begin{cases} init, & \text{if } l = \mathsf{self} \\ A, & \text{if } l = a \\ B, & \text{if } l = b \\ \text{undefined}, & \text{otherwise} \end{cases} \qquad \rho_4\, l \stackrel{\text{def}}{=} \begin{cases} A, & \text{if } l = a \\ B, & \text{if } l \in \{\mathsf{self}, b\} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

A process can write the tuple $f_1, \ldots, f_n$ in repository l – that is, the repository with address i, where i is the address which is bound to l by the allocation environment of the node where the process is running – by the output action $\mathbf{out}(f_1, \ldots, f_n)@l$. With an input action $\mathbf{in}(F_1, \ldots, F_n)@l$, a process can withdraw a datum that matches the pattern, or *template*, $(F_1, \ldots, F_n)$ from repository l. Standard pattern matching is used for templates and tuples.

With reference to our running example, suppose process *Boot*1 is defined as follows, where process *Boot*2 will be further specified later:

$$Boot1(!z, !w) \stackrel{\Delta}{=} (\mathbf{out}(GO)@z, d3).(\mathbf{out}(AF)@z, d3).(\mathbf{out}(BF)@w, d3).Boot2(z, w)$$

where the tokens *GO, AF, BF* $\in \mathcal{V}$ will be used for synchronising the components of the DMS and its clients. The following transition will result from the execution of the first action of process *Boot*1 in the last configuration we have considered above:

$$\begin{aligned} &init ::_{\rho_3} Boot1(a, b) \parallel A ::_{\rho_2} \mathbf{nil} \parallel B ::_{\rho_4} \mathbf{nil} \xrightarrow{(init, \mathbf{o}(GO, A)), d3} \\ &init ::_{\rho_3} (\mathbf{out}(AF)@a, d3).(\mathbf{out}(BF)@b, d3).Boot2(a, b) \parallel A ::_{\rho_2} \mathbf{nil} \parallel A ::_{\rho_2} \langle GO \rangle \parallel B ::_{\rho_4} \mathbf{nil} \end{aligned}$$

where a configuration is reached in which the token *GO* has been uploaded to site A. Incidentally, notice that the latter is represented by two nodes, one for processes, where currently **nil** is "in execution", and one for the token. It should be clear that as soon as the next two actions of *Boot*1 have been executed, the resulting configuration is the following one:

$$init ::_{\rho_3} Boot2(a, b) \parallel A ::_{\rho_2} \mathbf{nil} \parallel A ::_{\rho_2} \langle GO \rangle \parallel A ::_{\rho_2} \langle AF \rangle \parallel B ::_{\rho_4} \mathbf{nil} \parallel B ::_{\rho_4} \langle BF \rangle. \tag{3}$$

Processes can be written to/withdrawn from a repository as well. In particular, when a process is written to a remote repository, it loses the links of its localities to the addresses they are bound to by the local allocation environment; when the process will be (downloaded and) put into execution in a node, the allocation environment of *that* node will be used for resolving locality references occurring in the process. In other words, a *dynamic* scoping rule is used for the **out** operation. A *static* scoping discipline can be enforced by prefixing processes by an asterisk in the tuple-fields of the **out** operation. Action $\mathbf{read}(F_1, \ldots, F_n)@l$ is similar to $\mathbf{in}(F_1, \ldots, F_n)@l$, except that the datum at l is not deleted from the repository at l. The action $\mathbf{eval}(P)@l$ spawns process P at site l. Again, the *dynamic* scoping rule is used by default, while the *static* one can be enforced by the asterisk prefix (i.e. $\mathbf{eval}(*P)@l$). Notice that a locality variable u can be used in place of l in all above actions; we have used this possibility in our example above. It is worth pointing out that process *migration* can be modelled as *spawn & die*: suppose process Q, at a certain point of its behaviour, say after the execution of an action A, wants to move to locality l and continue there its execution according to the behaviour specified by Q'; then Q can be defined as follows:

$$Q \stackrel{\Delta}{=} \ldots.(A, r).(\mathbf{eval}(Q')@l, r').\mathbf{nil}.$$

Let us assume process *Boot*2 in our example is defined as follows:

$$Boot2(!v, !t) \stackrel{\Delta}{=} (\mathbf{eval}(*Srv(v, t))@v, d4).(\mathbf{eval}(*Usr(v))@v, d4).\mathbf{nil}$$

where process *Srv* models the behaviour of a service manager and process *Usr* that of users. The configuration reached right after the execution of *Boot*2*(a,b)* at *init* has terminated is the following:

$$\begin{aligned} &init ::_{\rho_3} \mathbf{nil} \parallel A ::_{\rho_2} \mathbf{nil} \parallel A ::_{\rho_2} \langle GO \rangle \parallel A ::_{\rho_2} \langle AF \rangle \parallel \\ &\quad A ::_{\rho_2} Srv(a, b)\{\rho_3\} \parallel A ::_{\rho_2} Usr(a)\{\rho_3\} \parallel B ::_{\rho_4} \mathbf{nil} \parallel B ::_{\rho_4} \langle BF \rangle \end{aligned} \tag{4}$$

$P\{\rho\}$ is the *closure* of process P with allocation environment ρ, and behaves like P except that any locality l in P denotes the physical address $(\rho\, l)$ if $l \in (\mathsf{dom}\, \rho)$, and is resolved with the current allocation environment otherwise.

Finally, we use the notation $\mathbf{busy}(r).P$ as a short hand for $(\mathbf{eval}(\mathbf{nil})@\mathsf{self}, r).P$, for any process P, whenever we want to model a delay with rate r, e.g. due to internal computation.

2.3. Tuples and templates

Tuple fields can be processes, localities, locality variables and value expressions. *Template fields* can be tuple fields, or *binders*, which are variables prefixed with an exclamation mark. The standard notion of free and bound occurrences of variables is used where binders indicate the binding occurrences of related variables and are associated to proper values by pattern-matching, e.g. when used in an **in** action.

2.4. StoKlaim specifications and their semantics

An StoKlaim system *specification* is a triple $(\beta_0, N_0, \vec{D})$ where $\beta_0 : \mathcal{R} \to \mathbb{R}_{>0}$ is a rate-mapping, N_0 and $\vec{D}$ are, respectively, a net modelling the initial configuration of the system and the process definitions for the processes used in N_0. For StoKlaim specification $\mathcal{S} = (\beta_0, N_0, \vec{D})$, we let $(\mathsf{Loc}\,\mathcal{S})$ and $(\mathsf{Adr}\,\mathcal{S})$ denote the set of localities and addresses, respectively, occurring in N_0 or $\vec{D}$. Notice that the above sets do not depend on β_0. With a little overloading, we use $(\mathsf{Loc}\,N)$ and $(\mathsf{Adr}\,N)$, for the network N as an abbreviation for $\mathsf{Loc}(\beta, N, \epsilon)$ and $\mathsf{Adr}(\beta, N, \epsilon)$, for any rate mapping β from rate-names to rates.

As we already mentioned earlier, the operational semantics definition of StoKlaim associates an LTS to each StoKlaim specification. For technical reasons, the states of the LTS are not just nets, but *configurations*, i.e. tuples (I, L, N), where $I \subseteq \mathcal{I}$ and $L \subseteq \mathcal{L}$ are the finite set of addresses and localities, respectively, in the net N. For configuration c, we let N_c (I_c and L_c respectively) denote the net (addresses and localities) component of c.

The *transition relation* $\Longrightarrow$ of StoKlaim is defined in [17] by means of a set of *reduction rules*, which make use of a *structural congruence* $\equiv$ on configurations defined by a set of *congruence laws*. The structural congruence essentially characterises commutativity, associativity and the neutral elements for network and process parallel composition as well as for choice; moreover, the structural congruence states the *cloning* principle, i.e.

$$(I, L, i ::_{\rho_1 \star \rho_2} P_1 \mid P_2) \equiv (I, L, i ::_{\rho_1} P_1 \parallel i ::_{\rho_2} P_2)$$

whenever ρ_1 and ρ_2 are *compatible*, i.e. $\rho_1\, l = \rho_2\, l$ if l belongs to both the domain of ρ_1 and the domain of ρ_2, in which case $\rho_1 \star \rho_2$ is defined as

$$(\rho_1 \star \rho_2)\, l \stackrel{\text{def}}{=} \begin{cases} \rho_1\, l & \text{if } l \in (\mathsf{dom}\,\rho_1) \\ \rho_2\, l & \text{if } l \in (\mathsf{dom}\,\rho_2). \end{cases}$$

In the following, we let $[c]$ denote the equivalence class of configurations c under $\equiv$ and $(rep\, c)$ denote the unique representative of $[c]$; we abstract here from the way these representatives are chosen. We let *RepCnf* denote the set of all representatives of the equivalence classes of configurations.

The StoKlaim transition relation definition given in [17] follows a similar pattern as that for Klaim in [16]: a *net transition relation* is defined which uses a lower level *process transition relation*; moreover a suitable rate-name *renaming* technique is introduced in order to distinguish different occurrences of the same rate-name, thus preserving the race condition principle. For configuration c, we let $(Der\ c)$ be the set of *derivatives* of c, i.e. the smallest set including c and all the configurations reachable from c via the StoKlaim transition relation. Finally, we let $(RDer\ c)$ be the set $\{rep(x) \mid x \in (Der\ c)\}$.

Notice that in [17], the specific technique used in order to preserve the race condition principle requires that the StoKlaim transition relation is a *parameterised* relation and that rate-name *strings* are used instead of just rate-names. In the present paper, we abstract from the details of the race condition preservation technique. Similar considerations apply to the function *Der*.

The LTS of a StoKlaim specification $\mathcal{S} = (\beta_0, N_0, \vec{D})$, denoted by $\mathsf{LTS}(\mathcal{S})$, is defined in the expected way. $\mathsf{LTS}(\mathcal{S})$ is the tuple $(C, \Lambda, \longrightarrow, c_0)$, where the initial state c_0 is $rep(((\mathsf{Adr}\,\mathcal{S}), (\mathsf{Loc}\,\mathcal{S}), N_0))$, i.e., the representative of (the congruence classes of) the configuration corresponding to the initial net N_0. The set C of states is $(RDer\, c_0)$, i.e. the set including c_0 and the representatives of (the congruence classes of) the configurations reachable from c_0 via the StoKlaim transition relation. Λ is the set of labels of the transitions; such labels are pairs of the form (γ, r). The

first component γ is an element of the set $\mathcal{I} \times \mathcal{A}$, where $\mathcal{A}$ is the set of *ground* actions, constructed according to the grammar below:

$$\mathcal{A} ::= \mathbf{n}(\mathcal{I}) \mid \mathbf{o}(\vec{\mathcal{F}}, \mathcal{I}) \mid \mathbf{i}(\vec{\mathcal{F}}, \mathcal{I}) \mid \mathbf{r}(\vec{\mathcal{F}}, \mathcal{I}) \mid \mathbf{e}(P, \mathcal{I})$$

for an output, input, read, eval, and newloc actions, respectively. The tuple parameters $\mathcal{F}$ are defined as follows:

$$\mathcal{F} ::= P \mid l \mid v.$$

The first component of γ is the address of the site where the action is executed, while the second component provides complete information on the action. For instance, $(i_1, \mathbf{o}(v, i_2))$ represents the uploading of value v from site i_1 to site i_2. The second component of the label of a transition is used in the mapping of the LTS to its associated CTMC. The transition relation $\rightarrow$ is defined in such a way that $c \xrightarrow{\gamma,r} c'$ if and only if there exists c'' such that $c \stackrel{\gamma,r}{\Longrightarrow} c''$ and $c' = (rep\, c'')$. The formal definition of $\mathsf{LTS}(\beta_0, N_0, \vec{D})$ follows:

Definition 1. For StoKlaim specification $\mathcal{S} = (\beta_0, N_0, \vec{D})$, $\mathsf{LTS}(\mathcal{S})$ is the tuple $(C, \Lambda, \longrightarrow, c_0)$, where

- $c_0 \stackrel{\text{def}}{=} (rep\, ((\mathsf{Adr}\,\mathcal{S}), (\mathsf{Loc}\,\mathcal{S}), N_0))$, is the *initial state*;
- $C \stackrel{\text{def}}{=} (RDer\, c_0)$ is the *set of states*;
- $c, (\gamma, r), c' \in \longrightarrow$ if and only if there exists c'' such that $c \stackrel{\gamma,r}{\Longrightarrow} c''$ and $c' = (rep\, c'')$;
- $\Lambda \stackrel{\text{def}}{=} \{(\gamma, r) \mid \exists c, c' \in C.\ (c, (\gamma, r), c') \in \longrightarrow\} \subseteq ((\mathcal{I} \times \mathcal{A}) \times \mathcal{R}))$ is the label-set.

We let $c \xrightarrow{\gamma,r} c'$ denote $(c, (\gamma, r), c') \in \longrightarrow$. In the sequel, we will consider only StoKlaim specifications with a finite LTS. It is worth pointing out here that moreover, the guardedness of process instantiation guarantees branching finiteness of the LTS generated from StoKlaim specifications.[2]

The translation of the LTS of a StoKlaim specification to a CTMC is fairly simple. Basically, rate-names need to be turned into rates. This entails that whenever $c \xrightarrow{\gamma,r} c'$ and $c \xrightarrow{\gamma,r'} c'$, a single γ-labelled transition from configuration c to c' should be obtained with rate $(\beta_0\, r) + (\beta_0\, r')$. In practice, we map LTSs to *action-labelled* CTMCs (AMCs), defined below:

Definition 2. An *action-labelled* CTMC (AMC) $\mathbb{A}$ is a triple $(S, \text{ACT}, \mapsto)$ where S is a set of states, ACT is a set of actions, and $\mapsto$ is the transition function, which is a total function from $S \times \text{ACT} \times S$ to the set of non-negative real numbers $\mathbb{R}_{\geq 0}$.

We use the notation $s \stackrel{\gamma,\lambda}{\longmapsto} s'$ whenever the transition function yields a positive value λ on (s, γ, s'). Transition $s \stackrel{\gamma,\lambda}{\longmapsto} s'$ intuitively means that the AMC may evolve from state s to s' while performing action γ with an execution time determined by an exponential distribution with rate λ. State *exit rates* and state *transition probabilities* are defined as expected:

Definition 3. For AMC $\mathbb{A}$ $(S, \text{ACT}, \mapsto)$, $s, s' \in S$ and $\gamma \in \text{ACT}$, the exit rate of s, $\mathbf{E}_{\mathbb{A}}(s)$, and the probability of moving from s to s' while performing action γ, $\mathbf{P}_{\mathbb{A}}(s, \gamma, s')$, are defined as follows:

$$\mathbf{E}_{\mathbb{A}}(s) \stackrel{\text{def}}{=} \sum_{s \stackrel{\gamma,\lambda}{\longmapsto} s'} \lambda$$

$$\mathbf{P}_{\mathbb{A}}(s, \gamma, s') \stackrel{\text{def}}{=} \begin{cases} \frac{\lambda}{\mathbf{E}_{\mathbb{A}}(s)} & \text{if there exists } \lambda > 0 \text{ such that } s \stackrel{\gamma,\lambda}{\longmapsto} s' \\ 0 & \text{otherwise.} \end{cases}$$

The finiteness of the AMC implies that $\mathbf{E}_{\mathbb{A}}(s)$ and $\mathbf{P}_{\mathbb{A}}(s, \gamma, s')$ are well defined.

The following definition characterises the AMC associated to a StoKlaim specification.

Definition 4. For StoKlaim specification $(\beta_0, N_0, \vec{D})$ with finite LTS $(C, \Lambda, \longrightarrow, c_0)$, let $\mathsf{AMC}(\beta_0, N_0, \vec{D}) \stackrel{\text{def}}{=} (S, \text{ACT}, \mapsto)$ with:

[2] There are several ways for assuring finiteness of transition systems obtained from process algebras; see, e.g., [24]. We will not dwell further upon this issue here.

Table 2
Process definitions for the DMS

$Boot$	$\triangleq$	$(\mathbf{newloc}(!x), d1).(\mathbf{newloc}(!y), d2).Boot1(x, y)$
$Boot1(!z, !w)$	$\triangleq$	$(\mathbf{out}(GO)@z, d3).(\mathbf{out}(AF)@z, d3).$ $(\mathbf{out}(BF)@w, d3).Boot2(z, w)$
$Boot2(!v, !t)$	$\triangleq$	$(\mathbf{eval}(*Srv(v, t))@v, d4).(\mathbf{eval}(*Usr(v))@v, d4).\mathbf{nil}$
$Usr(!l)$	$\triangleq$	$(\mathbf{busy}(urun)).UsrReq(l)$
$UsrReq(!l)$	$\triangleq$	$(\mathbf{in}(GO)@l, ur).UsrAct(l)$
$UsrAct(!l)$	$\triangleq$	$(\mathbf{out}(S1)@l, urs1).Usr(l) +$ $(\mathbf{out}(S2)@l, urs2).Usr(l)$
$Srv(!l, !r)$	$\triangleq$	$(\mathbf{in}(S1)@l, rs1).SrvAct1(l, r) +$ $(\mathbf{in}(S2)@l, rs2).SrvAct2(l, r)$
$SrvAct1(!l, !r)$	$\triangleq$	$(\mathbf{eval}(Agt1(l))@l, sa1).Srv(l, r)$
$SrvAct2(!l, !r)$	$\triangleq$	$(\mathbf{eval}(Agt2(l, r))@l, sa2).Srv(l, r)$
$SrvGo(!l)$	$\triangleq$	$(\mathbf{out}(GO)@l, sg).\mathbf{nil}$
$Agt1(!l)$	$\triangleq$	$(\mathbf{in}(AF)@l, gr1).Agt1Run(l)$
$Agt1Run(!l)$	$\triangleq$	$(\mathbf{busy}(a1run)).Agt1Done(l)$
$Agt1Done(!l)$	$\triangleq$	$(\mathbf{out}(AF)@l, a1done).SrvGo(l)$
$Agt2(!l, !r)$	$\triangleq$	$(\mathbf{in}(AF)@l, gr2).Agt2Run(l, r)$
$Agt2Run(!l, !r)$	$\triangleq$	$(\mathbf{busy}(a2runl)).Agt2Donel(l, r)$
$Agt2Donel(!l, !r)$	$\triangleq$	$(\mathbf{out}(AF)@l, a2donel).Agt2GetRs(l, r)$
$Agt2GetRs(!l, !r)$	$\triangleq$	$(\mathbf{in}(BF)@r, a2gr).Amr(l, r)$
$Amr(!l, !r)$	$\triangleq$	$(\mathbf{eval}(Agt2r(l, r))@r, amr).\mathbf{nil}$
$Agt2r(!l, !r)$	$\triangleq$	$(\mathbf{busy}(a2runr)).Agt2Doner(l, r)$
$Agt2Doner(!l, !r)$	$\triangleq$	$(\mathbf{out}(BF)@r, a2doner).SrvGo(l)$

- $S \stackrel{\text{def}}{=} C$
- $\text{ACT} \stackrel{\text{def}}{=} \{\gamma \in \mathcal{I} \times \mathcal{A} \mid \exists c, c', r.\ c \xrightarrow{\gamma, r} c'\}$
- $s \stackrel{\gamma,\lambda}{\longmapsto} s'$ if and only if $0 < \lambda = \sum_{s \xrightarrow{\gamma, r} s'} (\beta_0\ r)$.

For StoKlaim specification $(\beta_0, N_0, \vec{D})$, the underlying AMC has a unique initial distribution, viz. the one in which probability one is associated with the configuration corresponding to N_0, and zero with any other state.

We finally point out here that, in practice, the only place where β_0 plays a role is in the translation of $\mathsf{LTS}(\beta_0, N_0, \vec{D})$ to $\mathsf{AMC}(\beta_0, N_0, \vec{D})$. Consequently, one could alternatively define a StoKlaim specification as a *pair* $(N_0, \vec{D})$. This approach would be beneficial if one wants the actual rates depend on parameters other than just rate-names, like, for instance, the actual site addresses or data values involved in actions. All this information is obviously known only after the LTS has been computed. Since in the present paper rates depend only on rate-names, and rate-names are part of the specification, we defined a StoKlaim specification as a triple.

We close this section with the complete set of process definitions for the DMS example, which are given in Table 2. User requests for services are modelled by process *Usr*. This process repeatedly alternates between issuing a request for a service and being busy with other activities. In order to request a service, the user needs to obtain permission by means of a token *GO*, after which the user can issue a *S*1-type or *S*2-type request by placing the appropriate token on site *A*, referred to via locality *l*.

For each service request, the dispatcher process *Srv* spawns a specific agent on site *A* that will take care of serving it. This means that the agent needs to obtain the necessary resources, i.e. token *AF* for local resources and *BF* for

remote resources. Note that an agent taking care of an *S*2-type request, after having completed the local computation, acquires access to the remote resource, after which it migrates and runs remotely. Migration is modelled as spawn & die. Each agent terminates as soon as the processing of the request it is in charge of finishes. Before termination, it frees the resources it has used by reinserting the *AF* or *BF* token in the proper site(s) – agents for *S*2-type requests release the *AF* token as soon as they finish their local computation – as well as allowing further user requests to be issued by releasing the token *GO*.

We postpone the definition of the rate-mapping to Section 5, where we perform quantitative analysis via stochastic model-checking. We anticipate here that, in this example, we assume that communications are relatively faster than the computations by two orders of magnitude. This explicit separation of concerns, i.e. computation duration vs. communication duration, allows for the investigation of the effect of different assumptions concerning the stochastic behaviour of service components on the overall service performance.

3. The mobile stochastic logic MoSL$^+$

In order to enable the specification of the performance and dependability properties of StoKlaim processes, we propose to use a temporal logic. Given that the basic entities of our calculus are actions, the logic is both action- and state-based, as opposed to only state-based logics, such as LTL and CTL, and only action-based logics, such as ACTL. This entails that modal operators such as until are equipped with sets of actions. To be able to refer to the distributed character of the specified systems, the logic has some constructs to refer to the spatial nature of the system. These operators are inspired by the logic MoMo [22]. The spatial ingredients are embedded into the (action-based variant of the) real-time probabilistic logic CSL. This results in a logic with the following key features:

- it is a *temporal logic* that permits describing the dynamic evolution of the system;
- it is both *action-* and *state*-based;
- it is a *real-time* logic that permits the use of real-time bounds in the logical characterisation of the behaviours of interest;
- it is a *probabilistic logic* that permits expressing not only functional properties, but also properties related to performance and dependability aspects; and, finally
- it is a *spatial logic* that references the spatial structure of the network for the specification.

We start by presenting the syntax and semantics of MoSL$^+$, and then we consider the more practical issue of model-checking properties expressed in the logic.

3.1. Syntax

The syntactical definition of the logic makes use of all the basic syntactic categories introduced in Section 2. Additionally, the set $\mathcal{I}$-var of (physical) *address variables*, ranged over by $z, z', z_1, \ldots$, is used, and we let ι range over $\mathcal{I} \cup \mathcal{I}$-var.

3.1.1. Basic state formulae

Basic state formulae are built using a variant of the MoMo *consumption* ($\rightarrow$) and *production* ($\leftarrow$) operators. Production and consumption operators permit the formalisation of properties concerning the availability of resources (i.e. located tuples and processes) and system's reactions to the placement of new resources in a state.

Intuitively, a consumption formula

$$Q(\vec{Q}', \vec{\ell}, \vec{e})@\iota \rightarrow \Phi$$

holds for a network whenever in the network there exists a process Q running at a node, of site ι, and the "remaining" network, namely $Q(\vec{Q}', \vec{\ell}, \vec{e})$'s context, satisfies Φ. Notice that a process binder $!X$ can be used instead of process $Q(\vec{Q}', \vec{\ell}, \vec{e})$ and variable X can occur in Φ. Finally, instead of $Q(\vec{Q}', \vec{\ell}, \vec{e})$, a process variable X' can be used, which must be instantiated by means of an outer binder $!X'$, as we shall see in the sequel. Similarly, formula

$$\langle \vec{F} \rangle @\iota \rightarrow \Phi$$

holds whenever a tuple $\vec{f}$ matching $\vec{F}$ is stored in a node of site ι, and the "remaining" network satisfies Φ. The substitution resulting from pattern-matching is used to evaluate Φ. Basically, the consumption operator is a variant of the *tensor* operator of the Spatial Logic [10]. The components of the consumption are not restricted to processes, as it is the case for the tensor operator, and pattern-matching is provided.

With reference to our DMS example, it should be clear that the formula $Boot2(a, b)@init \rightarrow \mathtt{tt}$ (where $\mathtt{tt}$ is the constant *true*) holds of configuration (3), while configuration (1) does not satisfy it. Similarly, the formula $\langle AF\rangle@A \rightarrow \langle BF\rangle@B \rightarrow \mathtt{tt}$ is satisfied by configuration (3), but not by configuration (1). A typical use of the consumption operator is counting. For instance, the property *"two tuples, matching $\vec{f}$, are available at i"* is formalised as:

$$\langle \vec{f}\rangle@i \rightarrow \langle \vec{f}\rangle@i \rightarrow \mathtt{tt}.$$

Similarly, one can guarantee that "*there are at least three instances of process Q, two of which at site i_1 and the third one at site i_2*":

$$Q@i_1 \rightarrow Q@i_2 \rightarrow Q@i_1 \rightarrow \mathtt{tt}.$$

A production formula

$$Q(\vec{Q}', \vec{\ell}, \vec{e})@\iota \leftarrow \Phi$$

holds if the network satisfies Φ whenever process $Q(\vec{Q}', \vec{\ell}, \vec{e})$ is executed at (a node of) an existing site ι. Also, in this case a process variable X can be used instead (but not a binder). Similarly, the formula

$$\langle \vec{f}\rangle@\iota \leftarrow \Phi$$

holds if the network satisfies Φ whenever tuple $\vec{f}$ is stored in a node of existing site ι. The production operator resembles the *linear implication* associated to the tensor of [10]. Production formulae are very useful for context-system specifications. For instance, given a net N, one could be interested in studying the reaction of the system *if* a certain process Q is put in execution at site i of N, and in particular one may want to prove that the net still satisfies a certain property Φ, when Q is executed. This can be done by checking whether N satisfies $Q@i \leftarrow \Phi$. It is worth pointing out here that the property is checked over net N; in other words, the modeller does *not* need to modify the model, i.e. N, by adding Q, which is used in the *formula* only. This is the essence of context-verification.

In the case of DMS, productions can be used for specifying how the system reacts to a new service request. For instance, $\langle S2\rangle@A \leftarrow \Phi$ holds when Φ is satisfied after a $S2$-type service request is received. For instance, Φ could be used for specifying that a "*$S2$-type service execution is completed within t time units with probability that is at least q*".

We can summarise the grammar for basic state formulae as follows:

$$\aleph ::= \mathrm{PTF}@\iota \rightarrow \Phi \mid \langle \vec{F}\rangle@\iota \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@\iota \leftarrow \Phi \mid \langle \vec{f}\rangle@\iota \leftarrow \Phi$$

where *process template fields* PTF are defined according to the following grammar:

$$\mathrm{PTF} ::= Q(\vec{Q}', \vec{\ell}, \vec{e}) \mid !X.$$

We use the following abbreviations: $\mathrm{PTF}@\iota$ for $\mathrm{PTF}@\iota \rightarrow \mathtt{tt}$ and $\langle \vec{F}\rangle@\iota$ for $\langle \vec{F}\rangle@\iota \rightarrow \mathtt{tt}$. Recall that ι is either a physical address or an address variable. These variables are assigned values by means of pattern-matching against actual actions. Localities – in contrast to the modelling language – are *not* used for identifying sites in the logic. This is due to the fact that localities have a local connotation (which is resolved by local allocation environments), while at the property specification level one has a global view of the entire network. Like address variables, process variables occurring in the process template field under the scope of a binder introduced in action specifiers are also assigned values by means of pattern-matching action specifiers against actual actions.

3.1.2. Action specifiers and action sets

As in the branching-time temporal logic CTL, and also in MoSL$^+$ we distinguish between two classes of formulae, namely, *state* formulae Φ, Φ', Φ_1, ... and *path* formulae φ, φ', φ_1, ... As we deal with a combined state- and action-based model, it is useful to be able to refer to these actions in the logic, in much the same vein as in action-based

CTL [23]. In fact, the actions are specified by sets of *action specifiers*. For action specifier ξ_i, sets of action specifiers are built using the grammar:

$$\Delta ::= \top \mid \{\} \mid \{\xi_1, \ldots, \xi_n\}.$$

Here, $\top$ stands for "any set" and can be used when no requirement on actions is imposed. A set of action specifiers is satisfied by an action if the latter satisfies at least one of the elements of the set. Action specifiers are a kind of template for actions. They have the following shape:

$$\xi ::= g : \mathbf{N}(g) \mid g : \mathbf{O}(\vec{F}, g) \mid g : \mathbf{I}(\vec{F}, g) \mid g : \mathbf{R}(\vec{F}, g) \mid g : \mathbf{E}(\mathrm{PTF}, g)$$

where g is an address template, i.e., g is either of the form ι or $!z$. The action specifier $init : \mathbf{O}(GO, A)$, is satisfied only by action $(init, \mathbf{O}(GO, A))$. As we have seen in Section 2, the occurrence of this action models the uploading of value GO to site A by a process at site $init$. Action specifiers may contain binders that bind their variables to corresponding values in actions in the path; e.g., the action specifier $!z_1 : \mathbf{O}(GO, !z_2)$ is satisfied by any action, executed at some site, which uploads value GO to some site. This action specifier is satisfied, e.g. by action $(init, \mathbf{O}(GO, A))$. Action specifiers and their matching to actions generate substitutions in a natural way. The meanings of the other action specifiers are now self-explanatory.

3.1.3. Path formulae

The basic format of a path formula is the CTL *until* formula $\Phi\ \mathcal{U}\ \Psi$. In order to be able to refer also to actions executed along a path, we in fact use the variant of the *until* operator as originally proposed in action-based CTL [23]. To that end, the until-operator is parameterised with two action sets. A path satisfies $\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}\ \Psi$ whenever (eventually) a state satisfying Ψ – in the sequel, a Ψ-state – is reached via a Φ-path – i.e. a path composed only of Φ-states – *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ, and the Ψ-state is entered via an action satisfying Ω. Finally, we add a time constraint to path formulae. This is done by adding time parameter t – in much the same way as in timed CTL [1] – which is either a real number or may be infinite. In addition to the requirements described just above, it is now imposed that a Ψ-state should be reached within t time units. If $t = \infty$, this time constraint is vacuously true, and the until of action-based CTL is obtained. Similarly, a path satisfies $\Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Psi$ if the initial state satisfies Ψ (at time 0) or eventually a Ψ state will be reached in the path, by time t via a Φ-path, and, in addition, while evolving between Φ-states, actions are performed satisfying Δ. Accordingly, the syntax of path formulae is:

$$\varphi ::= \Phi\ {}_{\Delta}\mathcal{U}^{\leq t}_{\Omega}\ \Psi \mid \Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Psi.$$

Note that the only difference between the two until-operators is the absence or presence of the right-hand subscript, i.e., the action set specifying the constraints on the action which must be executed for entering the Ψ-state. We emphasise that $\Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Psi$ is *not* equivalent to $\Phi\ {}_{\Delta}\mathcal{U}^{\leq t}_{\top}\ \Psi$, because the latter formula requires that at least *one* transition is performed to reach a Ψ state, whereas this is not required in the former. The precise difference between the two until-formulae will become apparent when defining the semantics (cf. Section 3.2). Finally, notice that the above interpretation of the until-operators adheres to the standard interpretation of temporal logics. As we have seen, this entails that a formula $\Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Psi$ holds for a path whenever, e.g., the initial state satisfies Ψ. This should not be confused with "first passage" (and is also not meant to model this) where a transition into a Ψ-state is needed.

It should be easy to see that there is a computation in our running example starting from configuration (1) in Section 2.1 satisfying the formula

$$\mathtt{tt}\ {}_{\top}\mathcal{U}^{<\infty}_{init:\mathbf{O}(GO,A)}\ \mathtt{tt}$$

stating that eventually the token GO will be uploaded to site A from site $init$. Notice that the formula $\mathtt{tt}\ {}_{\top}\mathcal{U}^{<t}_{init:\mathbf{O}(GO,A)}\ \mathtt{tt}$, which also requires the action to be completed by time t, is substantially different from the following formula $\mathtt{tt}\ {}_{\top}\mathcal{U}^{\leq t}_{\top}\ \langle GO\rangle @ A$ which only states that GO must be present as the stored value at site A by time t (after at least one transition).

Obviously, variables may occur in formulae and are replaced by the associated values via the substitutions generated by action specifier pattern-matching. For example, $\mathtt{tt}\ {}_{\top}\mathcal{U}^{<\infty}_{i_1:\mathbf{N}(!z)}\ \mathbf{nil}@z$ states that a new node (referred to as) z is eventually going to be created from site i_1 and the **nil** process will be "running" there.

3.1.4. State formulae

Properties about states are formulated as state formulae. Basically, there are three categories of state formulae. The first category includes formulae in propositional logic, where the atomic propositions are tt and the basic state formulae introduced in Section 3.1.1. The second category includes statements about the likelihood of paths satisfying a property. Finally there are the so-called long-run properties. Of course, in general, a formula can be composed of sub-formulae of different categories. Let us be a bit more precise about the probabilistic path properties. Let φ be a property imposed on paths. State s satisfies the property $\mathcal{P}_{\bowtie p}(\varphi)$ whenever the total probability mass for all paths starting in s that satisfy φ meets the bound $\bowtie p$. Here, $\bowtie$ is a binary comparison operator from the set $\{<, >, \leq, \geq\}$, and p a probability in [0, 1]. For instance, the property $\mathcal{P}_{>0.99}(\mathit{legal}\ {}_{\top}\mathcal{U}_{\top}^{\leq 31.2}\ \mathit{goal})$ states that the probability to reach a goal state within 31.2 time units, via a path of legal states only, and with at least one transition, exceeds 0.99. Here, both the actions taken to move between legal states and the one for entering the goal state are irrelevant, as indicated by the action set $\top$. The following formula refers to the DMS model and states that if, in the current state, there is a request for a $S2$-type service placed on site A, the probability that this request gets served within 72, 04 time-units is at least 0.85 (the shorthand $\Phi \Rightarrow \Psi$ for $\neg\Phi \vee \Psi$ has been used):

$$\langle S2\rangle @ A \Rightarrow \mathcal{P}_{\geq 0.85}(\mathsf{tt}\ {}_{\top}\mathcal{U}^{<72,04}_{\{A:\mathbf{I}(S2,A)\}}\ \mathsf{tt}).$$

Long-run properties refer to the system when it has reached equilibrium. Under the assumption that the CTMC is finite, such an equilibrium will always exist [34]. A state s satisfies $\mathcal{S}_{\bowtie p}(\Phi)$ if, when starting from s, the probability of reaching a state which satisfies Φ in the long run $\bowtie p$. For instance, the formula

$$\mathcal{S}_{\geq 0.2}(\langle AF\rangle @ A)$$

states that, in the long run, the probability of finding the local resource free is at least 0.2. Interesting complex properties can be built by means of nesting the above operators. For instance, the following formula states that, in equilibrium, the probability is at least 0.87 that in at least 75% of the cases a $S1$-type request is placed at site A within 500 time units:

$$\mathcal{S}_{\geq 0.87}(\mathcal{P}_{\geq 0.75}(\mathsf{tt}\ {}_{\top}\mathcal{U}^{<500}_{\{!z:\mathbf{O}(S1,A)\}}\ \mathsf{tt})).$$

In summary, state-formulae are built according to the grammar:

$$\Phi \ ::= \ \mathsf{tt} \ \mid \ \aleph \ \mid \ \neg\Phi \ \mid \ \Phi \vee \Phi \ \mid \ \mathcal{P}_{\bowtie p}(\varphi) \ \mid \ \mathcal{S}_{\bowtie p}(\Phi).$$

3.2. Semantics

Paths play a central role in the formal definition of the semantics of MoSL$^+$. They are defined below:

Definition 5. Let $\mathbb{A} = (S, \mathrm{ACT}, \mapsto)$ be an action-labelled CTMC. A *path* π of $\mathbb{A}$ is a sequence

$$s_0\,(\gamma_0, t_0)\,s_1\,(\gamma_1, t_1)\ldots$$

such that the following two conditions hold:

- $s_j \in S$, $\gamma_j \in \mathrm{ACT}$, $t_j \in \mathbb{R}_{>0}$ and $s_j \overset{\gamma_j,\lambda}{\longmapsto} s_{j+1}$ for some $\lambda > 0$, for all $j \geq 0$;
- π is *maximal*, i.e. either it is *infinite* or there exists natural number l such that s_l is absorbing (i.e. there are no s, γ, and λ s.t. $s_l \overset{\gamma,\lambda}{\longmapsto} s$).

Five path operators which will be used in the sequel are defined in Table 3. They are $\mathsf{len}(\pi)$, giving the length of the path as the number of transitions it contains; $\mathsf{st}(\pi, j)$, giving the jth state in the path π; $\mathsf{ac}(\pi, j)$, giving the label of the jth transition in the path; $\mathsf{dl}(\pi, j)$, giving the actual delay of the jth transition if it exists in the path; $\pi(t)$, giving the state reached in the path after t time units passed. For any state s of an AMC $\mathbb{A}$, we let $\mathrm{Paths}_{\mathbb{A}}(s)$ denote the set of *all* paths $s_0(\gamma_0, t_0)s_1(\gamma_1, t_1)\ldots$ over $\mathbb{A}$ with $s_0 = s$. A Borel space can be defined over $\mathrm{Paths}_{\mathbb{A}}(s)$, together with its associated probability measure $\mathbb{P}$, which is a slight extension of that defined in [5] in order to take actions into consideration [19].

In the rest of the present paper, we assume formulae are *well-formed* w.r.t. a given StoKlaim specification $(\beta_0, N_0, \vec{D})$. Basically, a well-formed formula should not contain free variables, except those for which there is

Table 3
Operators on paths

For path $\pi = s_0\,(\gamma_0, t_0)\,s_1\,(\gamma_1, t_1)\ldots$, natural number j and $t \in \mathbb{R}_{\geq 0}$:

$$\mathsf{len}(\pi) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } \pi \text{ is infinite} \\ l & \text{otherwise , where } s_l \text{ is the absorbing state of } \pi \end{cases}$$

$$\mathsf{st}(\pi, j) \stackrel{\text{def}}{=} \begin{cases} s_j & \text{if } 0 \leq j \leq \mathsf{len}(\pi) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathsf{ac}(\pi, j) \stackrel{\text{def}}{=} \begin{cases} \gamma_j & \text{if } 0 \leq j < \mathsf{len}(\pi) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathsf{dl}(\pi, j) \stackrel{\text{def}}{=} \begin{cases} t_j & \text{if } 0 \leq j < \mathsf{len}(\pi) \\ \infty & \text{if } j = \mathsf{len}(\pi) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\pi(t) \stackrel{\text{def}}{=} \begin{cases} \mathsf{st}(\pi, \mathsf{len}(\pi)) & \text{if } t > \sum_{j=0}^{\mathsf{len}(\pi)-1} t_j \\ \mathsf{st}(\pi, m) & \text{otherwise, where } m = \min\{j \mid t \leq \sum_{k=0}^{j} t_k\}. \end{cases}$$

a defining equation in $\vec{D}$. Notice that action specifiers may introduce binding occurrences of variables: all free occurrences of a variable in Ψ are bound in $\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}^{<t}\ \Psi$ by a binder with the same name introduced in an action specifier which is element of set Ω. The formal definition of well-formedness can be found in [18].

3.2.1. Satisfaction relation

The following definition postulates when a StoKlaim specification satisfies an MoSL$^+$ formula.

Definition 6. A StoKlaim specification $(\beta_0, N_0, \vec{D})$ *satisfies* a state-formula Φ, written $(\beta_0, N_0, \vec{D}) \models_{SK} \Phi$ if and only if $s_0 \models_{\mathbb{A}} \Phi$, where s_0 is the state of $\mathbb{A} = \mathsf{AMC}(\beta_0, N_0, \vec{D})$ corresponding to the initial state c_0 of the LTS of $(\beta_0, N_0, \vec{D})$, as defined in Definition 4, and $\models_{\mathbb{A}}$ is defined in Table 5.

In the following sections, we will discuss the satisfaction relation $\models_{\mathbb{A}}$ for state formulae and path formulae. The latter will need the satisfaction relation defined for action specifiers as well.

3.2.2. State formulae

The satisfaction relation for state-formulae exploits pattern-matching. To that end, the definition of function match given in [17] must be extended in order to cover addresses; the complete definition is given in Table 4. There we use Θ to denote a substitution $[d_1/w_1 \ldots d_n/w_n]$, with $w_i \neq w_j$ for $i \neq j$, which replaces w_j by d_j for $0 < j \leq n$. Let [] denote the empty substitution and, w.l.o.g, for substitution Θ_1:

$$[d_1/w_1, \ldots, d_n/w_n, d'_1/w'_1, \ldots, d'_m/w'_m]$$

and substitution Θ_2:

$$[d''_1/w'_1, \ldots, d''_m/w'_m, d''_{m+1}/w'_{m+1}, \ldots, d''_{m+h}/w'_{m+h}]$$

with $\{w'_{m+1}, \ldots, w'_{m+h}\} \cap \{w_1, \ldots, w_n\} = \varnothing$, let $\Theta_1 \lhd \Theta_2$ be the substitution:

$$[d_1/w_1, \ldots, d_n/w_n, d''_1/w'_1, \ldots, d''_m/w'_m, d''_{m+1}/w'_{m+1}, \ldots, d''_{m+h}/w'_{m+h}].$$

Function match is a partial function which returns the substitution generated by matching its first argument, a template, against its second one, a tuple, when such a matching is successful.

Table 5 gives the definition of the satisfaction relation for MoSL$^+$ formulae. For deciding whether a state s satisfies formula $\mathcal{S}_{\bowtie p}(\Phi)$, the limit for $t \to \infty$ of the probability mass of the set of all those paths π starting from s and satisfying Φ at time t (i.e. $\pi(t) \models_{\mathbb{A}} \Phi$) must be computed and it must be checked whether it respects bound $\bowtie p$. Notice that such a limit always exists for *finite* CTMCs [34]. State s satisfies $\mathcal{P}_{\bowtie p}(\varphi)$ if the probability mass of the set of paths in $\mathbb{A}$ which satisfy φ is $\bowtie p$.

Table 4
Matching function

$$\mathrm{match}(l, l) \stackrel{\text{def}}{=} [] \qquad \mathrm{match}(v, v) \stackrel{\text{def}}{=} []$$

$$\mathrm{match}(!X, P\{\rho\}) \stackrel{\text{def}}{=} [P\{\rho\}/X] \qquad \mathrm{match}(!u, l) \stackrel{\text{def}}{=} [l/u] \qquad \mathrm{match}(!x, v) \stackrel{\text{def}}{=} [v/x]$$

$$\frac{\mathrm{match}(\vec{F}_1, \vec{f}_1) = \Theta_1 \qquad \mathrm{match}(\vec{F}_2, \vec{f}_2) = \Theta_2 \qquad \mathrm{match}(\vec{F}_3, \vec{f}_3) = \Theta_3}{\mathrm{match}(Q(\vec{F}_1, \vec{F}_2, \vec{F}_3), Q(\vec{f}_1, \vec{f}_2, \vec{f}_3)) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \Theta_2 \triangleleft \Theta_3}$$

$$\frac{\mathrm{match}(F_1, f_1') = \Theta_1 \qquad \ldots \qquad \mathrm{match}(F_n, f_n') = \Theta_n}{\mathrm{match}((F_1, \ldots, F_n), (f_1', \ldots, f_n')) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \cdots \triangleleft \Theta_n}$$

$$\mathrm{match}(i, i) \stackrel{\text{def}}{=} [] \qquad \mathrm{match}(!z, i) \stackrel{\text{def}}{=} [i/z]$$

$$\frac{\mathrm{match}(g_1, i_1) = \Theta_1 \qquad \mathrm{match}(\vec{F}, \vec{f}) = \Theta_2 \qquad \mathrm{match}(g_2, i_2) = \Theta_3}{\mathrm{match}(g_1 : \mathbf{O}(\vec{F}, g_2), (i_1, \mathbf{o}(\vec{f}, i_2))) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \Theta_2 \triangleleft \Theta_3}$$

$$\frac{\mathrm{match}(g_1, i_1) = \Theta_1 \qquad \mathrm{match}(\vec{F}, \vec{f}) = \Theta_2 \qquad \mathrm{match}(g_2, i_2) = \Theta_3}{\mathrm{match}(g_1 : \mathbf{I}(\vec{F}, g_2), (i_1, \mathbf{i}(\vec{f}, i_2))) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \Theta_2 \triangleleft \Theta_3}$$

$$\frac{\mathrm{match}(g_1, i_1) = \Theta_1 \qquad \mathrm{match}(\vec{F}, \vec{f}) = \Theta_2 \qquad \mathrm{match}(g_2, i_2) = \Theta_3}{\mathrm{match}(g_1 : \mathbf{R}(\vec{F}, g_2), (i_1, \mathbf{r}(\vec{f}, i_2))) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \Theta_2 \triangleleft \Theta_3}$$

$$\frac{\mathrm{match}(g_1, i_1) = \Theta_1 \qquad \mathrm{match}(F, P) = \Theta_2 \qquad \mathrm{match}(g_2, i_2) = \Theta_3}{\mathrm{match}(g_1 : \mathbf{E}(\mathrm{PTF}, g_2), (i_1, \mathbf{e}(P, i_2))) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \Theta_2 \triangleleft \Theta_3}$$

$$\frac{\mathrm{match}(g_1, i_1) = \Theta_1 \qquad \mathrm{match}(g_2, i_2) = \Theta_2}{\mathrm{match}(g_1 : \mathbf{N}(g_2), (i_1, \mathbf{n}(i_2))) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \Theta_2}$$

Table 5
Satisfaction relation for state formulae

$s \models_{\mathbb{A}} \mathsf{tt}$	
$s \models_{\mathbb{A}} \neg\, \Phi$	iff $s \not\models \Phi$ does not hold
$s \models_{\mathbb{A}} \Phi \vee \Psi$	iff $s \models_{\mathbb{A}} \Phi$ or $s \models_{\mathbb{A}} \Psi$
$s \models_{\mathbb{A}} \mathcal{S}_{\bowtie p}(\Phi)$	iff $\lim_{t\to\infty} \mathbb{P}\{\pi \in \mathrm{Paths}_{\mathbb{A}}(s) \mid \pi(t) \models_{\mathbb{A}} \Phi\} \bowtie p$
$s \models_{\mathbb{A}} \mathcal{P}_{\bowtie p}(\varphi)$	iff $\mathbb{P}\{\pi \in \mathrm{Paths}_{\mathbb{A}}(s) \mid \pi \models_{\mathbb{A}} \varphi\} \bowtie p$
$s \models_{\mathbb{A}} \mathrm{PTF}@i \to \Psi$	iff there exist N, ρ, P, and Θ s.t. the following *three* conditions hold: (1) $N_s \equiv N \parallel i ::_\rho P$ (2) $\mathrm{match}(\mathrm{PTF}, P) = \Theta$ (3) $(\beta_0, N, \vec{D}) \models_{SK} \Psi\Theta$
$s \models_{\mathbb{A}} \langle \vec{F} \rangle @i \to \Psi$	iff there exist N, ρ, $\vec{f}$, and Θ s.t. the following *three* conditions hold: (1) $N_s \equiv N \parallel i ::_\rho \langle \vec{f} \rangle$ (2) $\mathrm{match}(\vec{F}, \vec{f}) = \Theta$ (3) $(\beta_0, N, \vec{D}) \models_{SK} \Psi\Theta$
$s \models_{\mathbb{A}} Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Psi$	iff there exist N, E, and ρ s.t. the following *two* conditions hold: (1) $N_s \equiv N \parallel i ::_\rho E$ (2) $(\beta_0, N_s \parallel i ::_\rho Q(\vec{Q}', \vec{\ell}, \vec{e}), \vec{D}) \models_{SK} \Psi$
$s \models_{\mathbb{A}} \langle \vec{f} \rangle @i \leftarrow \Psi$	iff there exist N, E, and ρ s.t. the following *two* conditions hold: (1) $N_s \equiv N \parallel i ::_\rho E$ (2) $(\beta_0, N_s \parallel i ::_\rho \langle \vec{f} \rangle, \vec{D}) \models_{SK} \Psi$

In order for a state s to satisfy $\mathrm{PTF}@i \to \Psi$, its network component must contain a node $i ::_\rho P$, for some allocation environment ρ and process P which is matched by the process template field PTF; moreover, (the

Table 6
Satisfaction relation for action specifiers

γ, Θ	$\models_{\mathbb{A}} \top$	
γ, Θ	$\models_{\mathbb{A}} \{\xi_1, \ldots, \xi_n\}$	iff there exists $j, 0 < j \leq n$, s.t. $\gamma, \Theta \models_{\mathbb{A}} \xi_j$
γ, Θ	$\models_{\mathbb{A}} \xi$	iff $\text{match}(\xi, \gamma) = \Theta$

StoKlaim specification consisting of) the remaining network must satisfy Ψ, under the substitution generated by the pattern-matching.

The definition of the satisfaction relation for consumption formulae involving tuples is similar. In order for a state s to satisfy $Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Psi$, it is required that i is the address of an existing site in N_s; moreover N_s extended with the node $i ::_\rho Q(\vec{Q}', \vec{\ell}, \vec{e})$ must satisfy Ψ.

The definition of the satisfaction relation for the other kinds of state formulae is straightforward.

3.2.3. Sets of action specifiers

Table 6 gives the definition of the satisfaction relation for action specifiers and sets thereof. The concept behind the definition of the satisfaction relation for action specifiers is that an action γ satisfies an action specifier ξ if and only if the action *matches* the specifier, in which case a substitution is generated. Consequently, the satisfaction relation is defined over *(action, substitution)*-pairs and specifiers, and then extended to sets of action specifiers.

3.2.4. Path formulae

The definition of the satisfaction relation for path formulae, given in Table 7, formalises the meaning of the until operators, as discussed in Section 3.1.3. Notice that in the definition of $\Phi \; {}_\Delta\mathcal{U}_\Omega^{\leq t} \; \Psi$, the *only* substitution which is used for replacing variables with values is the one generated by the matching of the action of the *last* transition before the Ψ-state, and (an action specifier in) Ω; namely Θ_{k-1}. The bindings of all the previous, intermediate, substitutions are discarded. In this way, no counting or stack capability is included in the logic. Similar considerations apply to the simplified form of until, where *all* substitutions are indeed discarded Notice that, in this case also, the use of binders does make sense since they can be used as *don't care* placeholders.

3.2.5. Derived operators

Some frequently used operators can be derived from those of MoSL$^+$. The first set of derived operators, given on the left-hand-side of Table 8, shows how the standard until-operators from both action-based CTL and plain CTL are obtained, the next operator, and the modalities from Hennessy–Milner logic. The second set, given on the right-hand-side of the table, includes the eventually ($\Diamond$) and always ($\Box$) operators.

4. Model checking MoSL$^+$

In this section, we present a strategy for model checking MoSL$^+$ formulae against StoKlaim models. We introduce an algorithm that given a *finite* AMC $(S, \text{ACT}, \mapsto)$ generated from a StoKlaim specification $(\beta_0, N_0, \vec{D})$, and a MoSL$^+$ formula Φ, yields the states in S satisfying Φ.

Following a similar approach as that proposed in [29], model-checking of AMCs is performed by using a CSL model checker. In fact, the AMC to be model-checked is translated into an *equivalent* – in a sense which will be made clear in the sequel – state-labelled CTMC that can be analysed by making use of existing (state-based) CSL model checkers. In [18], we used a different approach, namely we translated (a fragment of) MoSL to aCSL, an action-based version of CSL, and we used an experimental version of the ETMCC model-checker which allows for action-based stochastic model-checking [19,18]. We stress here the fact that the MoSL to aCSL translation of [18] is not able to treat all forms of variable bindings, but its complexity is linear in both the size of the formula and the size of the AMC. The approach presented in the present paper covers the *full* logic MoSL$^+$, including the consumption and production operators and variable binding. However, as we will see, this comes at a potentially higher cost.

In the following, we first provide some auxiliary notions, after which we define the MoSL$^+$ model-checking algorithm and prove its correctness. In the rest of this section, given that the set of *states* of $\mathsf{AMC}(\beta_0, N_0, \vec{D})$ coincides with that of the LTS of $(\beta_0, N_0, \vec{D})$, we will use the words 'configuration' and 'state' as synonyms.

Table 7
Satisfaction relation for path formulae

$\pi \models_{\mathbb{A}} \Phi\ {}_{\Delta}\mathcal{U}^{<t}_{\Omega}\ \Psi$	iff there exists k, $0 < k \leq (\mathsf{len}\,\pi)$ s.t. the following *three* conditions hold: (1) $t > \sum_{j=0}^{k-1} \mathsf{dl}(\pi, j)$ (2) there exists Θ_{k-1} s.t. the following *three* conditions hold: (2.1) $\mathsf{st}(\pi, k-1) \models_{\mathbb{A}} \Phi$ (2.2) $\mathsf{ac}(\pi, k-1), \Theta_{k-1} \models_{\mathbb{A}} \Omega$ (2.3) $\mathsf{st}(\pi, k) \models_{\mathbb{A}} \Psi\Theta_{k-1}$ (3) if $k > 1$ then there exist $\Theta_0, \ldots, \Theta_{k-2}$ s.t. for all j, $0 \leq j \leq k-2$ the following *two* conditions hold: (3.1) $\mathsf{st}(\pi, j) \models_{\mathbb{A}} \Phi$ (3.2) $\mathsf{ac}(\pi, j), \Theta_j \models_{\mathbb{A}} \Delta$
$\pi \models_{\mathbb{A}} \Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Psi$	iff $\mathsf{st}(\pi, 0) \models_{\mathbb{A}} \Psi$ or there exists k, $0 < k \leq (\mathsf{len}\,\pi)$ s.t. the following *three* conditions hold: (1) $t > \sum_{j=0}^{k-1} \mathsf{dl}(\pi, j)$ (2) $\mathsf{st}(\pi, k) \models_{\mathbb{A}} \Psi$ (3) there exist $\Theta_0, \ldots, \Theta_{k-1}$ s.t. for all j, $0 \leq j \leq k-1$ the following *two* conditions hold: (3.1) $\mathsf{st}(\pi, j) \models_{\mathbb{A}} \Phi$ (3.2) $\mathsf{ac}(\pi, j), \Theta_j \models_{\mathbb{A}} \Delta$

Table 8
Derived operators

$\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}\ \Psi$	$\stackrel{\text{def}}{=}$	$\Phi\ {}_{\Delta}\mathcal{U}^{<\infty}_{\Omega}\ \Psi$	$\mathcal{P}_{\bowtie p}({}_{\Delta}\Diamond^{<t}_{\Delta'}\ \Phi)$	$\stackrel{\text{def}}{=}$	$\mathcal{P}_{\bowtie p}(\mathsf{tt}\ {}_{\Delta}\mathcal{U}^{<t}_{\Delta'}\ \Phi)$
$\Phi\ \mathcal{U}\ \Psi$	$\stackrel{\text{def}}{=}$	$\Phi\ {}_{\top}\mathcal{U}\ \Psi$	$\mathcal{P}_{\bowtie p}({}_{\Delta}\Box^{<t}_{\Delta'}\ \Phi)$	$\stackrel{\text{def}}{=}$	$\neg\mathcal{P}_{\bowtie p}({}_{\Delta}\Diamond^{<t}_{\Delta'}\ \neg\Phi)$
$\mathbf{X}^{<t}_{\Delta}\ \Phi$	$\stackrel{\text{def}}{=}$	$\mathsf{tt}\ {}_{\varnothing}\mathcal{U}^{<t}_{\Delta}\ \Phi$	$\mathcal{P}_{\bowtie p}({}_{\Delta}\Diamond^{<t}\ \Phi)$	$\stackrel{\text{def}}{=}$	$\mathcal{P}_{\bowtie p}(\mathsf{tt}\ {}_{\Delta}\mathcal{U}^{<t}\ \Phi)$
$\langle\Delta\rangle\ \Phi$	$\stackrel{\text{def}}{=}$	$\mathcal{P}_{>0}(\mathbf{X}_{\Delta}\ \Phi)$	$\mathcal{P}_{\bowtie p}({}_{\Delta}\Box^{<t}\ \Phi)$	$\stackrel{\text{def}}{=}$	$\neg\mathcal{P}_{\bowtie p}({}_{\Delta}\Diamond^{<t}\ \neg\Phi)$
$[\Delta]\ \Phi$	$\stackrel{\text{def}}{=}$	$\neg\langle\Delta\rangle\neg\Phi$			

In the definition of the model-checking algorithm, we will often have to manipulate the input AMC obtained from an StoKLAIM specification, in particular when dealing with the production and consumption operators of the logic. The results of such manipulations may be AMCs which cannot be obtained from any particular StoKLAIM specification alone, but are obtained from a set of representative configurations by means of the StoKLAIM reduction rules and congruence laws. Therefore, we first extend Definition 4 in order to obtain an AMC not only from a single specification, but also from a set of representatives.

Let sets $\mathcal{I}$, $\mathcal{R}$, $\mathcal{A}$ and *RepCnf* be the sets of addresses, rate-names, ground actions and configuration (congruence classes) representatives, as introduced in Section 2, together with the StoKLAIM transition relation $\Longrightarrow$ and set *RDer*.

Definition 7. A set $C \subseteq \mathit{RepCnf}$ is StoKLAIM-closed if and only if whenever $c \in C$ and $c \stackrel{\gamma,r}{\Longrightarrow} c'$, also $(\mathit{rep}\ c') \in C$.

It is easy to see that, for any finite set $C \subset \mathit{RepCnf}$, the set $(\mathit{RDer}\ C)$, defined – with a little bit of overloading – as $\bigcup_{c \in C} \mathit{RDer}\ c$, is a StoKLAIM-closed set.

Definition 8. For rate-mapping β, $C \subseteq \mathit{RepCnf}$ and process definitions $\vec{D}$, we let $\mathsf{AMC}(\beta, S, \vec{D})$ denote the AMC $\mathbb{A} = (S, \mathrm{ACT}, \mapsto)$ such that:

- $S = (\mathit{RDer}\ C)$
- $\mathrm{ACT} = \{\gamma \in \mathcal{I} \times \mathcal{A} \mid \exists s, s' \in S, r \in \mathcal{R}.\ s \stackrel{\gamma,r}{\Longrightarrow} s'\}$
- $s \stackrel{\gamma,\lambda}{\longmapsto} s'$ if and only if $0 < \lambda = \sum_{s \stackrel{\gamma,r}{\Longrightarrow} s'} (\beta\ r)$.

In the sequel, for $\mathsf{AMC}(\beta, C, \vec{D}) = \mathbb{A} = (S, \mathrm{ACT}, \mapsto)$, and node element E, we say that E *is a component of* $\mathbb{A}$, written $E \preceq \mathbb{A}$ if and only if there exists $s \in S, i \in I_s, N$ and ρ such that $N_s \equiv N \parallel i ::_\rho E$.

Let N be a net, i an address of a node in N, and E a node element; we let $N \oplus (i, E)$ be the net obtained from N by adding element E at existing address i. Similarly, $N \ominus (i, E)$ denotes the net obtained from N by removing existing element E from i. For instance, $i ::_\rho \langle 3\rangle \ominus (i, \langle 3\rangle) = i ::_\rho \mathbf{nil}$, $i ::_\rho \mathbf{nil} \oplus (i, P) = i ::_\rho P$, while $i ::_\rho \langle 3\rangle \ominus (i, \langle 4\rangle)$ and $i_1 ::_\rho P \oplus (i_2, Q)$ are undefined. The formal definition of $\oplus$ and $\ominus$ follows. $N \oplus (i, E)$ is defined only if the argument net N has a node with address i, in which case $i ::_{[\mathsf{self}\to i]} E$ is added to N with address i and with the minimal allocation environment $[\mathsf{self} \to i]$. We take the minimal allocation environment in order not to make any assumptions on it. The StoKlaim semantics [17] takes care of the proper aggregation of the allocation environments of different nodes modelling together a site (i.e. nodes with the same address). The definition of $\ominus$ is similar, except that now a node element is removed from an existing node with address i. Please notice that when a component is removed, the allocation environment in the corresponding site is not modified; this guarantees that name resolution is preserved after an element has been removed.

Definition 9. For net N, address i and node element E:

$$N \oplus (i, E) \stackrel{\text{def}}{=} \begin{cases} N \parallel i ::_{[\mathsf{self}\to i]} E, & \text{if } N \equiv N' \parallel i ::_\rho E' \text{ for some } N', \rho, E' \\ \text{undefined}, & \text{otherwise} \end{cases}$$

$$N \ominus (i, E) \stackrel{\text{def}}{=} \begin{cases} N' \parallel i ::_\rho \mathbf{nil}, & \text{if } N \equiv N' \parallel i ::_\rho E \text{ for some } N', \rho \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The same operators can be applied to representative configurations. Let $c \in \mathit{RepCnf}$; $c \oplus (i, E)$ and $c \ominus (i, E)$ denote the representative configurations obtained from c, by adding or removing node E to/from i, respectively:

Definition 10. For $c \in \mathit{RepCnf}$, $C \subseteq \mathit{RepCnf}$, address i and node element E:

$$c \oplus (i, E) \stackrel{\text{def}}{=} \begin{cases} (\mathit{rep}\, c'), & \text{if } \exists N.\, N = N_c \oplus (i, E) \text{ and } c' = (I_c \cup (\mathsf{Adr}\, N), L_c \cup (\mathsf{Loc}\, N), N) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

$$c \ominus (i, E) \stackrel{\text{def}}{=} \begin{cases} (\mathit{rep}\, c'), & \text{if } \exists N.\, N = N_c \ominus (i, E) \text{ and } c' = (I_c \cup (\mathsf{Adr}\, N), L_c \cup (\mathsf{Loc}\, N), N) \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The above operators are lifted to sets of representative configurations as follows:

$$C \oplus (i, E) \stackrel{\text{def}}{=} \{c \oplus (i, E) \mid c \in C\}$$
$$C \ominus (i, E) \stackrel{\text{def}}{=} \{c \ominus (i, E) \mid c \in C\}.$$

Similar operators can be defined for $\mathsf{AMC}(\beta, C, \vec{D})$.

Definition 11. Let $\mathsf{AMC}(\beta, C, \vec{D}) = (S, \mathrm{ACT}, \mapsto)$, $\mathsf{AMC}(\beta, C, \vec{D}) \oplus (i, E)$ and $\mathsf{AMC}(\beta, C, \vec{D}) \ominus (i, E)$ are defined as follows:

$$\mathsf{AMC}(\beta, C, \vec{D}) \oplus (i, E) = \mathsf{AMC}(\beta, S \oplus (i, E), \vec{D})$$
$$\mathsf{AMC}(\beta, C, \vec{D}) \ominus (i, E) = \mathsf{AMC}(\beta, S \ominus (i, E), \vec{D}).$$

It is important to point out here that $\mathbb{A} \oplus (i, E)$ may be infinite even if $\mathbb{A}$ is finite, because node element E could in principle be an infinite process, like, e.g. OL where $OL \stackrel{\Delta}{=} (\mathbf{out}(\langle v\rangle)@\mathsf{self}, r).OL$. We underline that the results shown in the rest of the present section, and in particular the (termination and) correctness of the model-checking algorithm, are valid only under the assumption that all intermediate AMCs computed by the model-checking algorithm using $\ominus$ and $\oplus$ are finite.

On the basis of the above defined operators, we define an alternative characterisation of the satisfaction relation for production and consumption formulae as follows, where $\mathbb{A} = \mathsf{AMC}(\beta, C, \vec{D})$ for some β, C, and $\vec{D}$:

Definition 12.

- $s \models_{\mathbb{A}} \mathrm{PTF}@i \to \Psi$ if and only if there exists $P \preceq \mathbb{A}$ such that $\mathrm{match}(\mathrm{PTF}, P) = \Theta$ and $s \ominus (i, P) \models_{\mathbb{A}\ominus(i,P)} \Psi\Theta$
- $s \models_{\mathbb{A}} \vec{F}@i \to \Psi$ if and only if there exists $\vec{f} \preceq \mathbb{A}$ such that $\mathrm{match}(\vec{F}, \vec{f}) = \Theta$ and $s \ominus (i, \vec{f}) \models_{\mathbb{A}\ominus(i,\vec{f})} \Psi\Theta$

- $s \models_{\mathbb{A}} Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Psi$ if and only if $s \oplus (i, Q(\vec{Q}', \vec{\ell}, \vec{e})) \models_{\mathbb{A} \oplus (i, Q(\vec{Q}', \vec{\ell}, \vec{e}))} \Psi$
- $s \models_{\mathbb{A}} \vec{f}@i \leftarrow \Psi$ if and only if $s \oplus (i, \vec{f}) \models_{\mathbb{A} \oplus (i, \vec{f})} \Psi$.

In order to perform model-checking of a finite AMC $\mathbb{A}$, we translate $\mathbb{A}$ into a finite, state-labelled, CTMC, which can be analysed using existing stochastic model-checkers. The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently, the CTMCs we consider are essentially state-labelled ones, where the labelling function is implicit. We first recall the definition of (finite) CTMC.

Definition 13. A finite CTMC $\mathbb{M}$ is a tuple pair $(S, \mathbf{R})$ with S a finite set of states and $\mathbf{R}$ the rate matrix, i.e. a total function in $S \times S \to \mathbb{R}_{\geq 0}$.

A CTMC can be viewed as an AMC where each transition label consists of the transition rate only. Hence, Definitions 3 and 5 can be easily extended to cover CTMCs.

We use a similar mapping from AMCs to CTMCs as in [29] which essentially moves action labels from transitions to the states that those transitions are pointing to. For each state s in the source AMC, and for each transition pointing to s labelled by an action, say a, and a rate, say λ, a distinct duplicate of s, labelled by a, with incoming transition labelled by λ is created in the target CTMC. Moreover, in order to consider the first transition delay correctly, one additional $\perp$-labelled duplicate is added for s. The outgoing transitions of these duplicate states have the same targets and same rates as those of the original state. This means that, by construction, all copies of state s in the target CTMC are strong Markovian bisimilar and therefore enjoy the same transient and steady state properties [29]. The translation is formally defined as follows, where $\mathbb{A} = \mathsf{AMC}(\beta, C, \vec{D})$ for some β, C, and $\vec{D}$:

Definition 14. For finite AMC $\mathbb{A} = (S, \text{ACT}, \mapsto)$ let $\mathcal{K}(\mathbb{A})$ be the CTMC $(S', \mathbf{R})$ such that:

- $S' \stackrel{\text{def}}{=} \{(s, \gamma) \mid s \in S \wedge \exists s' \in S, \lambda > 0.\ s' \stackrel{\gamma,\lambda}{\longmapsto} s\} \cup \{(s, \perp) \mid s \in S\}$
- $\mathbf{R}((s', \gamma'), (s, \gamma)) \stackrel{\text{def}}{=} \begin{cases} \lambda & \text{if } \exists \lambda.\ s' \stackrel{\gamma,\lambda}{\longmapsto} s \\ 0 & \text{otherwise.} \end{cases}$

Notice that, for each path $\pi = s_0(\gamma_0, t_0)s_1(\gamma_1, t_1)s_2 \ldots$ of AMC $\mathbb{A}$, there is a unique path $\mathcal{K}(\pi) \stackrel{\text{def}}{=} (s_0, \perp)t_0(s_1, \gamma_0)t_1(s_2, \gamma_1) \ldots$ in $\mathcal{K}(\mathbb{A})$. For set X of paths in $\mathbb{A}$, we let $\mathcal{K}(X)$ denote the set $\{\mathcal{K}(\pi) \mid \pi \in X\}$.

Furthermore, our model-checking algorithm relies on the functions until and steady defined as follows:

Definition 15. For CTMC $\mathcal{M} = (S, \mathbf{R})$, $S_1, S_2 \subseteq S$, $t \in \mathbb{R}_{\geq 0}$, $p \in [0, 1]$, and $\bowtie \in \{<, >, \leq, \geq\}$:

$$\text{until}(\bowtie, p, t, S_1, S_2, \mathcal{M}) \stackrel{\text{def}}{=} \{s \in S \mid \mathbb{P}\{\pi \in \text{Paths}(s) \mid \exists t' < t.\pi(t') \in S_2 \text{ and } \forall t'' < t'.\pi(t'') \in S_1\} \bowtie p\}$$

$$\text{steady}(\bowtie, p, S_1, \mathcal{M}) \stackrel{\text{def}}{=} \{s \in S \mid \lim_{t \to \infty} \mathbb{P}\{\pi \in \text{Paths}(s) \mid \pi(t) \in S_1\} \bowtie p\}.$$

Please notice that both until and steady can be computed using a CSL model checker when interpreting the sets $S1$ and $S2$ as the sets of states that satisfy the sub-formulae occurring in the until and steady state formulae.

We now introduce some lemmas that we will use later for proving the soundness and completeness of the proposed model checker. Their proofs are the same as those of analogous lemmas in [29], and they state a number of important relationships between the finite StoKlaim AMC and the CTMCs which they are mapped to.

Lemma 16. *For AMC $\mathbb{A} = (S, ACT, \mapsto)$, $s, s' \in S$ the following three statements hold:*

- *$\pi \in Paths_{\mathbb{A}}(s)$ if and only if $\mathcal{K}(\pi) \in Paths_{\mathcal{K}(\mathbb{A})}((s, \perp))$;*
- *for all $x \in ACT \cup \{\perp\}$, $\mathbf{E}_{\mathbb{A}}(s) = \mathbf{E}_{\mathcal{K}(\mathbb{A})}(s, x)$;*
- *for all $x, \gamma \in ACT \cup \{\perp\}$, $\mathbf{P}_{\mathbb{A}}(s, \gamma, s') = \mathbf{P}_{\mathcal{K}(\mathbb{A})}((s, x), (s', \gamma))$.*

The first statement says that there is a one-to-one correspondence between paths in $\mathbb{A}$ starting in s and those in $\mathcal{K}(\mathbb{A})$ starting in $(s, \perp)$. The second statement relates the total exit rate of states in $\mathbb{A}$ to that of corresponding states in $\mathcal{K}(\mathbb{A})$. The third statement says that the probability of moving from state s to s' with action γ in $\mathbb{A}$ is equal to the probability of moving from state (s, x) to state (s', γ) in $\mathcal{K}(\mathbb{A})$ for any action $x \in \text{ACT} \cup \{\perp\}$.

Lemma 17. *For AMC $\mathbb{A} = (S, ACT, \mapsto)$, $s \in S$ and measurable set $X \subseteq Paths_{\mathbb{A}}(s)$, the following holds: $\mathbb{P}\{X\} = \mathbb{P}\{\mathcal{K}(X)\}$.*

The above lemma states that corresponding sets of paths in both structures have the same probability mass.

The next lemma states two properties that concern steady state probabilities. The first statement says that the steady state probability of strong Markovian bisimilar states in $\mathcal{K}(\mathbb{A})$ are equal. The second statement says that the steady state probability of a state s in $\mathbb{A}$ is equal to the steady state probability of the state $(s, \bot)$ in the transformed CTMC $\mathcal{K}(\mathbb{A})$. This is due to the construction of $\mathcal{K}(\mathbb{A})$.

Lemma 18. *For AMC $\mathbb{A} = (S, ACT, \mapsto)$, $s \in S$, $x, y \in ACT \cup \{\bot\}$, and $X \subseteq S$, the following holds:*

- $\lim_{t\to\infty} \mathbb{P}\{\pi \in Paths_{\mathcal{K}(\mathbb{A})}(s, x) \mid \exists z.\ \pi(t) = (s', z) \text{ and } s' \in X\} = \lim_{t\to\infty} \mathbb{P}\{\pi \in Paths_{\mathcal{K}(\mathbb{A})}(s, y) \mid \exists z.\ \pi(t) = (s', z) \text{ and } s' \in X\}$
- $\lim_{t\to\infty} \mathbb{P}\{\pi \in Paths_{\mathbb{A}}(s) \mid \pi(t) \in X\} = \lim_{t\to\infty} \mathbb{P}\{\pi \in Paths_{\mathcal{K}(\mathbb{A})}(s, \bot) \mid \exists z.\ \pi(t) = (s', z) \text{ and } s' \in X\}$.

Proof. It is easy to see that, by construction, paths that start in states that only differ in their action labels are identical except for the label of their first state, but this does not influence the steady state probability of such set.

Moreover, thanks to Lemmas 16 and 17, we have that for each t:

- $\mathbb{P}\{\pi \in \text{Paths}_{\mathbb{A}}(s) \mid \pi(t) \in X\} = \mathbb{P}\{\pi \in \text{Paths}_{\mathcal{K}(\mathbb{A})}(s, \bot) \mid \exists z.\ \pi(t) = (s', z) \text{ and } s' \in X\}$. □

4.1. Model-checking algorithm

In this section, a model-checking algorithm is given for MoSL$^+$ which uses functions until and steady, defined on standard, i.e. *non* action-based CTMCs. Notice also that the algorithm is able to deal with binding variables in MoSL$^+$ formulae.

Definition 19. For rate-mapping β, finite $C \subseteq RepCnf$, process definitions $\vec{D}$, $\mathbb{A} = (S, \text{ACT}, \mapsto) = \mathsf{AMC}(\beta_0, C, \vec{D})$, and the MoSL$^+$ formula Φ, $\text{Sat}(\Phi, \mathbb{A})$ returns the set of all states of $\mathbb{A}$ which satisfy Φ, and is defined recursively on the structure of Φ as follows:

- $\text{Sat}(\mathtt{tt}, \mathbb{A}) \stackrel{\text{def}}{=} S$
- $\text{Sat}(\neg\, \Phi, \mathbb{A}) \stackrel{\text{def}}{=} S \setminus \text{Sat}(\Phi, \mathbb{A})$
- $\text{Sat}(\Phi \vee \Psi, \mathbb{A}) \stackrel{\text{def}}{=} \text{Sat}(\Phi, \mathbb{A}) \cup \text{Sat}(\Psi, \mathbb{A})$
- $\text{Sat}(\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}^{<t}_{\Omega}\ \Psi), \mathbb{A}) \stackrel{\text{def}}{=}$
  let $X = \{\gamma \mid \gamma \in \text{ACT} : \exists\Theta.\text{match}(\gamma, \Delta) = \Theta\}$ in
  let $Y = \{(\gamma, \Theta) \mid \gamma \in \text{ACT} : \text{match}(\gamma, \Omega) = \Theta\}$ in
  let $S_1 = \text{Sat}(\Phi, \mathbb{A}) \times (X \cup \{\bot\})$ in
  let $S_2 = \bigcup_{(\gamma,\Theta)\in Y} \text{Sat}(\Psi\Theta, \mathbb{A}) \times \{\gamma\}$ in
  $\{s \in S \mid (s, \bot) \in \text{until}(\bowtie, p, t, S_1, S_2, \mathcal{K}(\mathbb{A}))\}$
- $\text{Sat}(\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Psi), \mathbb{A}) \stackrel{\text{def}}{=}$
  let $X = \{\gamma \in \text{ACT} \mid \exists\Theta.\text{match}(\gamma, \Delta) = \Theta\}$ in
  let $S_1 = \text{Sat}(\Phi, \mathbb{A}) \times (X \cup \{\bot\})$ in
  let $S_2 = \text{Sat}(\Psi, \mathbb{A}) \times (X \cup \{\bot\})$ in
  $\{s \in S \mid (s, \bot) \in \text{until}(\bowtie, p, t, S_1, S_2, \mathcal{K}(\mathbb{A}))\}$
- $\text{Sat}(\mathcal{S}_{\bowtie p}(\Phi), \mathbb{A}) \stackrel{\text{def}}{=}$
  $\{s \in S \mid (s, \bot) \in \text{steady}(\bowtie, p, \text{Sat}(\Phi, \mathbb{A}) \times (\text{ACT} \cup \{\bot\}), \mathcal{K}(\mathbb{A}))\}$
- $\text{Sat}(\text{PTF}@i \to \Psi, \mathbb{A}) \stackrel{\text{def}}{=}$
  let $X = \{(P, \Theta) \mid P \preceq \mathbb{A} \text{ and } \text{match}(\text{PTF}, P) = \Theta\}$ in
  $\bigcup_{(P,\Theta)\in X} \{s \in S \mid s \ominus (i, P) \in \text{Sat}(\Psi\Theta, \mathbb{A} \ominus (i, P))\}$

- $\mathrm{Sat}(\langle \vec{F} \rangle @ i \to \Psi, \mathbb{A}) =$
  let $X = \{(\vec{f}, \Theta) \mid \vec{f} \preceq \mathbb{A} \text{ and } \mathrm{match}(\vec{F}, \vec{f}) = \Theta\}$ in
  $$\bigcup_{(\vec{f},\Theta)\in X} \{s \in S \mid s \ominus (i, \vec{f}) \in \mathrm{Sat}(\Psi\Theta, \mathbb{A} \ominus (i, \vec{f}))\}$$
- $\mathrm{Sat}(P @ i \leftarrow \Psi, \mathbb{A}) = \{s \mid s \oplus (i, P) \in \mathrm{Sat}(\Psi, \mathbb{A} \oplus (i, P))\}$
- $\mathrm{Sat}(\langle \vec{f} \rangle @ i \leftarrow \Psi, \mathbb{A}) = \{s \mid s \oplus (i, \vec{f}) \in \mathrm{Sat}(\Psi, \mathbb{A} \oplus (i, \vec{f}))\}$.

We briefly address the non-trivial cases of the above definition of $\mathrm{Sat}(\Phi, \mathbb{A})$.

The set of states in $\mathbb{A}$ that satisfy the formula $\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}^{<t}_{\Omega}\ \Psi)$ is computed by applying CSL model-checking on the CTMC $\mathcal{K}(\mathbb{A})$ of CSL-until-formula $\mathcal{P}_{\bowtie p}(\alpha_1\ \mathcal{U}^{<t}\ \alpha_2)$, where α_j is a fresh new atomic proposition labelling all and only those states belonging to set S_j, for $j = 1, 2$. S_1 is computed by taking the set of states in $\mathbb{A}$ that satisfy Φ (applying function $\mathrm{Sat}(\Phi, \mathbb{A})$ recursively) and pairing these states with those actions that satisfy constraint Δ (i.e. set X). The set S_2 is computed in a similar way, but taking also possible bindings into consideration that derive from the matching of actions with the constraint Ω and that may bind variables in Ψ. This aspect is taken care of by the recursive application of $\mathrm{Sat}(\Psi\Theta, \mathbb{A})$ on the MoSL$^+$ formula $\Psi\Theta$ and $\mathbb{A}$.

Computing the set of states satisfying $\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Psi)$ is similar, but simpler, because there is no constraint on the actions of those transitions that reach a state that satisfies Ψ.

Computing the set of states satisfying a steady state formula reduces to a simple steady state analysis on the transformed CTMC, considering those states of the form $(s, \bot)$.

At this point, there are four cases left that concern the consumption and production operators. We first deal with the consumption operator. We are looking for those states in $\mathbb{A}$ that satisfy Ψ assuming that we have removed a process P from a node in $\mathbb{A}$ with address i that satisfies the process template field PTF. First, the set X of processes is computed that appear in $\mathbb{A}$, and that match PTF generating the substitution Θ. Then the set of states satisfying the consumption formula is the union over those sets of states s such that, given pair $(P, \Theta) \in X$, once P is removed from s, the state satisfies the formula Ψ with substitution Θ in the structure $\mathbb{A}$ without P at i. The approach for computing the set of states satisfying a formula containing the consumption operator for all tuples of data is similar.

The satisfaction of a production formula requires that the satisfaction of formula Ψ is verified on $\mathbb{A}$ extended with process P at address i (if it exists), or tuple f at address i (if it exists).

The correctness of the algorithm is stated by Theorem 20, followed by a formal proof.

Theorem 20. *Given* StoKlaim *specification* $(\beta_0, N_0, \vec{D})$ *with AMC* $\mathbb{A} = (S, ACT, \mapsto) \stackrel{\text{def}}{=} \mathsf{AMC}(\beta_0, N_0, \vec{D})$*, for each* $s \in S$ *and* MoSL$^+$ *formula* Φ*,* $s \in \mathrm{Sat}(\Phi, \mathbb{A})$ *if and only if* $s \models_{\mathbb{A}} \Phi$.

Proof. The proof is given along the same lines as the proof of Theorem 5 in [29]. By induction on the structure of Φ, where the Induction Hypothesis can be formulated as:

For each β, $C \subseteq \mathit{RepCnf}$ and $\vec{D}$ such that $\mathbb{A} = \mathsf{AMC}(\beta, C, \vec{D})$, on all MoSL$^+$ formulas Ψ of length smaller than Φ, it holds that $s \in \mathrm{Sat}(\Psi, \mathbb{A})$ iff $s \models_{\mathbb{A}} \Psi$.

Case Φ of:

tt:

trivial, since, for all $s \in S$ we have that $s \models_{\mathbb{A}} \mathsf{tt}$, by definition of $\models_{\mathbb{A}}$ and $s \in \mathrm{Sat}(\mathsf{tt}, \mathbb{A})$, by definition of Sat.

$\rightleftharpoons$

$\neg\Phi$ and $\Phi \vee \Psi$:

Directly follow from the Induction Hypothesis.

$\rightleftharpoons$

$\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}^{<t}_{\Omega}\ \Psi)$:

We define the set $A^s_n(t)$ of the paths π in $\mathrm{Paths}_{\mathbb{A}}(s)$ which satisfy $\Phi\ {}_{\Delta}\mathcal{U}^{<t}_{\Omega}\ \Psi$, and such that the n-th state of π is a Ψ-state and it is reached by time t via a Ω-transition, while all previous states are Φ-states and only Δ-transitions are performed.

Let:

$$X = \{\gamma \in \mathrm{ACT} \mid \exists\Theta.\mathrm{match}(\gamma, \Delta) = \Theta\}$$

$Y = \{(\gamma, \Theta) \mid \gamma \in \mathrm{ACT} : \mathrm{match}(\gamma, \Omega) = \Theta\}$
$S_1 = \{s \mid s \models_{\mathbb{A}} \Phi\}$
$S_2 = \{(s, \gamma) \mid \exists \Theta.(\gamma, \Theta) \in Y \text{ and } s \models_{\mathbb{A}} \Psi\Theta\}$.

Please notice that X and Y are computable, because ACT is finite; moreover, S_1 and S_2 are computable using function Sat, due to the Induction Hypothesis. For any $n \geq 1$, $s \in S$ and $t > 0$, we let $A_n^s(t)$ be the set of paths defined as follows:

$$A_n^s(t) \stackrel{\text{def}}{=} \left\{ \pi \in \mathrm{Paths}_{\mathbb{A}}(s) \;\middle|\; \sum_{j=0}^{n-1} \mathsf{dl}(\pi, j) < t \wedge (\mathsf{st}(\pi, n), \mathsf{ac}(\pi, n-1)) \in S_2 \right.$$
$$\left. \wedge \forall 0 \leq k \leq n-1 . \mathsf{st}(\pi, k) \in S_1 \wedge \forall 0 \leq k < n-1 . \mathsf{ac}(\pi, k) \in X \right\}.$$

It should be clear, from the construction of $A_n^s(t)$, that if $\pi \in A_n^s(t)$ then $\pi \models_{\mathbb{A}} \Phi \; {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \; \Psi$.

We now define set $B_n^s(t)$ denoting the subset of $A_n^s(t)$ consisting of paths that reach a Ψ-state for the first time in exactly n steps, within t time units, i.e. without performing any transition satisfying both Δ and Ω leading to a Ψ-state in a previous step. In other words, those paths that reach a Ψ-state for the first time exactly at the n-th step.

For any $n \geq 1$, $s \in S$ and $t > 0$, we let $B_n^s(t)$ be the set of paths defined recursively on n as follows:

$$B_1^s(t) \stackrel{\text{def}}{=} A_1^s(t);$$
$$B_{i+1}^s(t) \stackrel{\text{def}}{=} A_{i+1}^s(t) \setminus \bigcup_{j=1}^{i} B_i^s(t).$$

It is easy to see that for all $i \neq j$, $B_i^s(t) \cap B_j^s(t) = \varnothing$. This allows for the summation of their probabilities given the underlying Borel space construction.

Moreover, from the definition of $B_i^s(t)$ it follows that if a path $\pi \in \mathrm{Paths}_{\mathbb{A}}(s)$ satisfies $\Phi \; {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \; \Psi$, then there exists i such that $\pi \in B_i^s(t)$.

This allows us to compute the probabilities of the paths satisfying the until-formula $\Phi \; {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \; \Psi$ as the sum of the probabilities of the paths that reach a Ψ-state for the first time in i steps within time t and for $i \in \mathbb{N}$:

$$\mathbb{P}\{\pi \in \mathrm{Paths}_{\mathbb{A}}(s) \mid \pi \models_{\mathbb{A}} \Phi \; {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \; \Psi\} = \mathbb{P} \bigcup_{i=0}^{\infty} B_i^s(t) = \sum_{i=0}^{\infty} \mathbb{P} B_i^s(t) \tag{5}$$

Similarly, for any $n \geq 1$, $s \in S$ and $t > 0$ we let $\hat{A}_n^{(s,\perp)}(t)$ be the set of paths in $\mathcal{K}(\mathbb{A})$ defined as follows:

$$\hat{A}_n^{(s,\perp)}(t) = \left\{ \pi \in \mathrm{Paths}_{\mathcal{K}(\mathbb{A})}(s, \perp) \;\middle|\; \sum_{j=0}^{n-1} \mathsf{dl}(\pi, j) < t \wedge \mathsf{st}(\pi, n) \in S_2 \right.$$
$$\left. \wedge \forall 0 \leq k \leq n-1 . \mathsf{st}(\pi, k) \in S_1 \times (X \cup \{\perp\}) \right\}.$$

Moreover, for any $n \geq 1$, $s \in S$ and $t > 0$, we let $\hat{B}_n^{(s,\perp)}(t)$ be the set of paths defined recursively on n as follows:

$$\hat{B}_1^{(s,\perp)}(t) = \hat{A}_1^{(s,\perp)}(t);$$
$$\hat{B}_{i+1}^{(s,\perp)}(t) = \hat{A}_{i+1}^{(s,\perp)}(t) \setminus \bigcup_{j=1}^{i} \hat{B}_i^{(s,\perp)}(t).$$

By the construction of $\hat{B}_i^{(s,\perp)}(t)$, we get

$$\mathbb{P}\{\pi \in \mathrm{Paths}(s) \mid \exists t' \leq t . \pi(t') \in S_2 \text{ and } \forall t'' \leq t' . \pi(t'') \in S_1 \times (X \cup \{\perp\})\}$$
$$= \mathbb{P} \bigcup_{i=0}^{\infty} \hat{B}_i^{(s,\perp)}(t) = \sum_{i=0}^{\infty} \mathbb{P} \hat{B}_i^{(s,\perp)}(t) \tag{6}$$

It is easy to show that, for all $i \geq 1$,

$$\mathcal{K}(B_i^s(t)) = \hat{B}_i^{(s,\perp)}(t). \tag{7}$$

We can now proceed with the following derivation: $s \models_{\mathbb{A}} \mathcal{P}_{\bowtie p}(\Phi \ {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \ \Psi)$

$\equiv$ {Def. of $\models_{\mathbb{A}}$}

$\mathbb{P}\{\pi \in \mathrm{Paths}_{\mathbb{A}}(s) \mid \pi \models_{\mathbb{A}} \varphi\} \bowtie p$

$\equiv$ {Equation (5) above}

$\sum_{i=0}^{\infty} \mathbb{P}B_i^s(t) \bowtie p$

$\equiv$ {Lemma 17}

$\sum_{i=0}^{\infty} \mathbb{P}\mathcal{K}(B_i^s(t)) \bowtie p$

$\equiv$ {Equation (7) above}

$\sum_{i=0}^{\infty} \mathbb{P}\hat{B}_i^{(s,\perp)}(t) \bowtie p$

$\equiv$ {Def. of until; Equation (6)}

$(s, \perp) \in \mathrm{until}(\bowtie, p, t, S_1 \times (X \cup \{\perp\}), S_2, \mathcal{K}(\mathbb{A}))$

$\equiv$ {Def. of Sat}

$s \in \mathrm{Sat}(\mathcal{P}_{\bowtie p}(\Phi \ {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \ \Psi), \mathbb{A})$

$\rightleftharpoons$

$\mathcal{P}_{\bowtie p}(\Phi \ {}_{\Delta}\mathcal{U}^{<t} \ \Psi)$:
Similar to the case $\mathcal{P}_{\bowtie p}(\Phi \ {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \ \Psi)$

$\rightleftharpoons$

$\mathcal{S}_{\bowtie p}(\Phi)$:
directly from Lemma 18.

$\rightleftharpoons$

$\mathrm{PTF}@i \to \Psi$:

$s \models_{\mathbb{A}} \mathrm{PTF}@i \to \Psi$

$\equiv$ {Def. of $\models_{\mathbb{A}}$}

$\exists N, \rho, P, \Theta.\ N_s \equiv N \parallel i ::_{\rho} P \wedge \mathrm{match}(\mathrm{PTF}, P) = \Theta \wedge (\beta_0, N, \vec{D}) \models_{SK} \Psi\Theta$

$\equiv$ {Def. of $\ominus$, $\preceq$}

$\exists i, P, \Theta.\ P \preceq \mathbb{A} \wedge \mathrm{match}(\mathrm{PTF}, P) = \Theta \wedge s \ominus (i, P) \models_{\mathbb{A}\ominus(i,P)} \Psi\Theta$

$\equiv$ {Induction Hypothesis}

$\exists i, P, \Theta.\ P \preceq \mathbb{A} \wedge \mathrm{match}(\mathrm{PTF}, P) = \Theta \wedge s \ominus (i, P) \in \mathrm{Sat}(\Psi\Theta, \mathbb{A} \ominus (i, P))$

$\equiv$ {Def. of Sat}

$s \in \mathrm{Sat}(\mathrm{PTF}@i \to \Psi, \mathbb{A})$

$\rightleftharpoons$

$\langle \vec{F} \rangle @i \to \Psi$:
Similar to the case $\mathrm{PTF}@i \to \Psi$

$\rightleftharpoons$

$Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Psi$:
$s \models_{\mathbb{A}} Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Psi$

$\equiv$ {Def. of $\models_{\mathbb{A}}$}

$\exists N, E, \rho.\ N_s \equiv N \parallel i ::_\rho E \wedge (\beta_0, N_s \parallel i ::_\rho Q(\vec{Q}', \vec{\ell}, \vec{e}), \vec{D}) \models_{SK} \Psi$

$\equiv$ {Def. of $\oplus$}

$\exists i.\ s \oplus (i, Q(\vec{Q}', \vec{\ell}, \vec{e})) \models_{\mathbb{A} \oplus (i, Q(\vec{Q}', \vec{\ell}, \vec{e}))} \Psi$

$\equiv$ {Induction Hypothesis}

$\exists i.\ s \oplus (i, Q(\vec{Q}', \vec{\ell}, \vec{e})) \in \mathrm{Sat}(\Psi, \mathbb{A} \oplus (i, Q(\vec{Q}', \vec{\ell}, \vec{e})))$

$\equiv$ {Def. of Sat}

$s \in \mathrm{Sat}(Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Psi, \mathbb{A})$

$\rightleftharpoons$

$\langle \vec{f} \rangle @i \leftarrow \Psi$:
Similar to the case $Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Psi$. □

4.2. Time complexity

Let $(\beta_0, N_0, \vec{D})$ be an StoKlaim specification, and $\mathbb{A} = \mathrm{AMC}(\beta_0, N_0, \vec{D})$. If N_0 is composed by n processes of size k, then $\mathbb{A}$ will contain $\mathcal{O}(n^k)$ states. Similarly, to compute $\mathbb{A} \oplus (i, E)$ (respectively $\mathbb{A} \ominus (i, E)$) we need, in the worst case, $\mathcal{O}(n^k)$ steps.

Let Φ be a MoSL$^+$ formula and $\mathbb{A} = \mathrm{AMC}(\beta_0, N_0, \vec{D})$. To compute $\mathrm{Sat}(\Phi, \mathbb{A})$, if Φ does not contain *consumption* and *production* operators, we need (just as for other CSL model checkers) a number of steps that is polynomial in the size of $\mathbb{A}$ and linear in the length of Φ. When considering formulae with consumption and production, the computation of $\mathrm{Sat}(\Phi, \mathbb{A})$ would need a number of steps polynomial in n^k.

To decrease the complexity of the algorithm, we are considering to investigate an on-the-fly approach to stochastic model checking that could avoid generating all the possible reachable configurations.

4.3. Front-end tool

The proposed algorithm has been implemented in a prototype tool, named SAM, for supporting the analysis of mobile and distributed systems specified using StoKlaim. SAM allows one to simulate StoKlaim specifications and to generate their reachability graphs. Moreover, the tool permits the verification of whether a StoKlaim specification satisfies an MoSL$^+$ formula.

The core of the system, which is implemented in OCaml, consists of two components: `stocklaimgraph` and `moslmc`. The first component permits the analysis of the execution of StoKlaim programs and generating their reachability graphs. The second one, after loading an StoKlaim net and a formula, verifies, by means of one or more calls to the MRMC model checker, the satisfaction of the formula by the net. MRMC, which is implemented in `C`, is used for computing function steady and until defined in Definition 15. OCaml supports a basic mechanism for interoperability with the `C` language. This permits OCaml code to call `C` functions.

5. Analysis of the DMS example

In this section, we show how MoSL$^+$ can be used to specify integrated qualitative and quantitative properties of (models described by) StoKlaim specifications. We also give the results of model-checking some such MoSL$^+$ formulae against StoKlaim models by means of the model-checking algorithm described in Section 4 using SAM and the MRMC model-checker. We make explicit reference to our DMS running example.

As we pointed out in Section 2, the explicit separation of concerns, computation time vs. communication time, allows for the investigation of the impact of different assumptions concerning the stochastic behaviour of service components on the overall service performance.

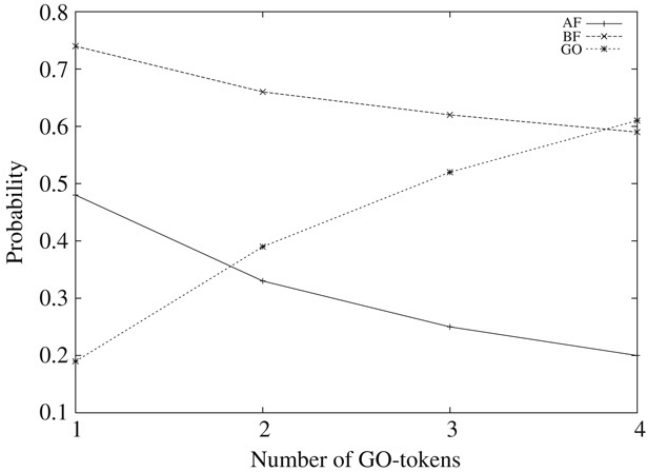

Fig. 1. Resource occupation in presence of multiple *GO*-tokens.

We analysed the DMS example under the assumption that the average duration of all internal computations (of the user and of the agents in charge of services) is 0.20 time units, while the average duration of communications is two orders of magnitude shorter, i.e. 0.002 time units. So, our assumption is that the services of the DMS are computationally heavy, while the communication infrastructure is very efficient. The complete specification of our DMS example is thus the tuple $(\beta_0, N_0, \vec{D})$. The process definitions $\vec{D}$ are those given in Table 2; the initial net is $init ::_{[\mathsf{self} \to init]} Boot$, and the rate-mapping β_0 is defined as follows:

$$\beta_0\, r \stackrel{\text{def}}{=} \begin{cases} 5, & \text{if } r \in \{urun, a1run, a2runl, a2runr\} \\ 500, & \text{otherwise.} \end{cases}$$

The rates assigned have been selected only for the purposes of illustrating the analyses, and do not reflect any existing system.

Fig. 1 shows the resource use of the DMS example under various assumptions on the capability of the system to pipeline (buffer) the user requests. This can be modelled by varying the number of *GO*-tokens in the DMS specification, allowing the user to issue new requests before previous ones have been served; alternatively, this can be expressed by means of the production operator. The resource use of the local (A) and remote (B) site can be derived from the long-term probability of states that reflect the presence of the AF and BF tokens in the local and remote sites respectively. The curves in Fig. 1 show the long term probability of the local resource being free (AF present at site A), the remote resource being free (BF present at site B), and the system being able to accept user requests (GO present at site A) for a number of initial GO tokens ranging from 1 to 4. The steady-state probability of the local resource being free can be easily obtained by model-checking the formula $\mathcal{S}_{\geq 0.2}(\langle AF\rangle @ A)$. In fact, the model-checker, besides returning the set of states which satisfy the formula, provides also the actual probability to be, in the long run, in a $\langle AF\rangle @ A$-state, i.e. the probability that, in the long run, the token AF is present at site A. In a similar way, the long term probabilities related to BF and GO can be computed.

For the DMS system, one could be also interested in studying responsiveness to service requests. For instance, the following property can be considered: "Whenever a request for an $S2$-type service is issued at site A while the remote resource has been continuously free, the probability that the related service is taken care of at site B within time t is at least p". This property can be rendered in MoSL$^+$ by the following formula:

$$\Phi \stackrel{\Delta}{=} \mathcal{P}_{\leq 0}(\langle BF\rangle @ B\ {}_{\top}\mathcal{U}_{\{A:\mathbf{O}(S2,A)\}}\ \neg\mathcal{P}_{>p}({}_{\top}\Diamond^{<t}_{\{A:\mathbf{E}(Agt2r,B)\}}\ \mathsf{tt})).$$

We analysed the system for different values of t and p. The following table contains some possible values of t and p which guarantee the satisfaction of Φ after the boot phase.

Time (t)	0.1	0.3	0.5	0.7	0.9
Prob. ($p\geq$)	0.355	0.763	0.912	0.967	0.988

For instance, Φ is satisfied if $t = 0.5$ while p is greater or equal to 0.912. The same formula is not satisfied if one considers $t = 0.7$ and $p < 0.967$.

Further examples of interesting properties of the DMS that can be expressed in the logic MoSL$^+$ are the following:

- If there is a request for a $S2$-type service, then the probability that this request gets served within t time-units is at least p.

  $$\langle S2\rangle @ A \Rightarrow \mathcal{P}_{\geq p}(\mathsf{tt}\ {}_{\top}\mathcal{U}^{<t}_{\{A:\mathbf{I}(S2,A)\}}\ \mathsf{tt}).$$

  This formula is satisfied by the initial configuration of the DMS: No $S2$-type service request is available when the system starts.
- If the remote resource fails with a rate fr when it is not in use and it can be repaired with a rate rr, the probability is smaller than p that after performing a $S2$-type request (within t time units), a state can be reached in which the probability of observing the evaluation of $Agt2r$ on the remote site within t' is smaller than q:

  $$Fail@B \leftarrow \mathcal{P}_{<p}({}_{\top}\Diamond^{<t}_{\{A:\mathbf{O}(S2,A)\}}\ \mathcal{P}_{<q}({}_{\top}\Diamond^{<t'}_{\{A:\mathbf{E}(Agt2r,B)\}}\ \mathsf{tt}))$$

  where *Fail* and *Repair* are the agents defined as follows:

  $$\begin{array}{lcl} Fail & \stackrel{\Delta}{=} & (\mathbf{in}(BF)@b,\ fr).Repair \\ Repair & \stackrel{\Delta}{=} & (\mathbf{out}(BF)@b, rr).Fail. \end{array}$$

  Agent *Fail* models the possibility for the resource in B to fail (when it is not in use) with a *failure rate* fr.
  Agent *Repair* models the capability of the system to recover from a failure of the resource in B with a *recover rate* of rr.
  We extend the rate mapping β_0 in such a way that $\beta_0(fr) = 1.0$ while $\beta_0(rr) = 10.0$. Under this assumptions, we have that the DMS satisfies the formula above when $t' = 1.0$, $t = 0.3$ and $p = q = 0.5$. The same formula is not satisfied when we let $t = 0.4$ while let other values unchanged.
- The probability is greater than p that a $S2$-type service execution is completed within t time units while it is guaranteed that a request for a $S1$-type service gets served within t' time-units with probability that is at least q:

  $$\langle GO\rangle @ A \rightarrow \langle S2\rangle @ A \leftarrow \mathcal{P}_{>p}((\Phi)\ {}_{\top}\mathcal{U}^{<t}_{\{B:\mathbf{O}(GO,A)\}}\ \mathsf{tt})$$

  where:

  $$\Phi \stackrel{\Delta}{=} S1@A \leftarrow \mathcal{P}_{>q}({}_{\top}\Diamond^{<t'}_{\{A:\mathbf{I}(S1,A)\}}\ \mathsf{tt}).$$

  Using the tool, we have verified that the formula above is not satisfied by the DMS when $t = t' = 0.3$ and $p = q = 0.8$. The same formula is satisfied when we increase the value of t to 0.4.

6. Conclusions and future work

This paper presented the temporal Mobile Stochastic Logic, MoSL$^+$, which permits the description of both action and state-based properties of mobile concurrent systems. The logic is aimed at the formal specification of performance and dependability properties of systems modelled with StoKlaim, a process description language that can model mobile, distributed systems, and associates to process actions a random duration. The logic addresses both spatial and temporal notions in order to reflect the topological structure of systems and their evolution over time in a probabilistic setting. MoSL$^+$, is a branching-time temporal logic (a la CTL) in which the universal and existential path quantifiers have been replaced by a probability operator, in the same vein as in PCTL [3,15] and CSL [2].

For path-formulae, the until-operator is the main modality, and is decorated with sets of actions that describe the possible activities along the path, just like the action-based variant of CTL presented in [4] and with time-bounds like the timed CTL presented in [1]. These operators naturally express steady-state probabilities as well as probability measures of paths. Specific atomic actions are used to characterize relevant activities taking place during executions, and continuous random variables with exponential distributions are used for modelling action durations. Sets of actions are captured by simple formulae that may contain variables to be bound to actions occurrences; this mechanism allows, for instance, to deal with the dynamic creation of sites by processes. State formulae permit, instead, the capturing of spatial aspects of systems by formalizing properties concerning resources availability and resources consumption, by means of production and consumption operators similar to those of the MoMo logic [22].

We have presented the syntax and formal semantics of MoSL$^+$; due to the subtle interplay between real-time, probabilistic, and spatial aspects, the semantics is somewhat technically involved. The theoretical achievements have been complemented by a model-checking algorithm for MoSL$^+$. To illustrate the approach, a small example modelling a distributed service which incorporates agent mobility has been considered, and the analysis of some of its qualitative and quantitative aspects has been performed.

The results in this paper show the viability of the approach and its practical usefulness when addressing the quantitative aspects of mobile systems. We plan to further assess our proposal by considering more challenging case studies. At the linguistic level, we plan to investigate the extension of MoSL$^+$ and its model checking algorithm with rewards, using again the functionality provided by MRMC. To decrease the complexity of the current model checking algorithm, we shall try to adopt an on-the-fly approach that should avoid generating all possible reachable configurations. We plan also to investigate richer and more dynamic network structures and to study the associated modalities to be used instead of the current flat set of localities. A candidate starting point is a calculus with explicit links and explicit operations for node connections and disconnections similar to OPEN-KLAIM [6]. Other extensions could be considered to capture properties of allocation environments explicitly.

Acknowledgments

We would like to thank the anonymous reviewers for fruitful comments that enabled us to improve the presentation of our work.

References

[1] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science 126 (1994) 183–235.
[2] A. Aziz, K. Sanwal, V. Singhal, R. Brayton, Model checking continuous time Markov chains, ACM Transactions on Computational Logic 1 (1) (2000) 162–170.
[3] A. Aziz, V. Singhal, F. Balarin, It usually works: The temporal logic of stochastic systems, in: Proceedings of CAV 1995, in: LNCS, vol. 939, 1995, pp. 155–165.
[4] C. Baier, L. Cloth, B. Haverkort, M. Kuntz, M. Siegle, Model checking action- and state-labelled Markov chains, in: 2004 International Conference on Dependable Systems & Networks, IEEE Computer Society Press, ISBN: 0-7695-2052-9, 2004, pp. 701–710.
[5] C. Baier, J.-P. Katoen, H. Hermanns, Approximate symbolic model checking of continuous-time Markov chains, in: J. Baeten, S. Mauw (Eds.), Concur '99, in: LNCS, vol. 1664, Springer-Verlag, 1999, pp. 146–162.
[6] L. Bettini, V. Bono, R. De Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, B. Venneri, The Klaim project: Theory and practice, in: C. Priami (Ed.), Global Computing: Programming Environments, Languages, Security and Analysis of Systems, in: LNCS, vol. 2874, Springer-Verlag, 2003, pp. 88–150.
[7] L. Bettini, R. De Nicola, M. Loreti, Formalizing properties of mobile agent systems, in: F. Arbab, C. Talcott (Eds.), Coordination Models and Languages, in: LNCS, vol. 2315, Springer-Verlag, 2002, pp. 72–87.
[8] A. Bianco, L. de Alfaro, Model checking of probabalistic and nondeterministic systems, in: Proceedings of FSTTCS 1995, in: LNCS, vol. 1026, 1995, pp. 499–513.
[9] G. Boudol, ULM: A core programming model for global computing: (Extended abstract), in: D.A. Schmidt (Ed.), Programming Languages and Systems, 13th European Symposium on Programming, ESOP, in: LNCS, vol. 2986, Springer-Verlag, 2004, pp. 234–248.
[10] L. Caires, L. Cardelli, A spatial logic for concurrency (part I), Information and Computation 186 (2) (2003) 194–235.
[11] L. Cardelli, A language with distributed scope, in: 22nd Annual ACM Symposium on Principles of Programming Languages, ACM, 1995, pp. 286–297.
[12] L. Cardelli, Abstractions for mobile computations, in: J. Vitek, C. Jensen (Eds.), Secure Internet Programming, in: LNCS, vol. 1603, Springer-Verlag, 1999, pp. 51–94.
[13] L. Cardelli, A. Gordon, Anytime, anywhere: Modal logics for mobile ambients, in: Twentyseventh Annual ACM Symposium on Principles of Programming Languages, ACM, 2000, pp. 365–377.
[14] G. Castagna, J. Vitek, Seal: A framework for secure mobile computations, in: H. Bal, B. Belkhouche, L. Cardelli (Eds.), Internet Programming Languages, in: LNCS, vol. 1686, Springer-Verlag, 1999, pp. 47–77.
[15] L. de Alfaro, Temporal logics for the specification of performance and reliability, in: Proceedings of STACS 1997, in: LNCS, vol. 1200, 1997, pp. 165–176.
[16] R. De Nicola, G. Ferrari, R. Pugliese, KLAIM: A kernel language for agents interaction and mobility, IEEE Transactions on Software Engineering. IEEE CS 24 (5) (1998) 315–329.
[17] R. De Nicola, J.-P. Katoen, D. Latella, M. Loreti, M. Massink, KLAIM and its stochastic semantics, Technical Report, Dipartimento di Sistemi e Informatica, Università di Firenze. Available at: http://rap.dsi.unifi.it/~loreti/papers/TR062006.pdf, 2006.
[18] R. De Nicola, J.-P. Katoen, D. Latella, M. Loreti, M. Massink, MoSL: A stochastic logic for STOKLAIM, Technical Report, Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo'. Available at: http://www1.isti.cnr.it/~Latella/MoSL.pdf, 2006.

[19] R. De Nicola, J.-P. Katoen, D. Latella, M. Massink, Towards a logic for performance and mobility, in: A. Cerone, H. Wiklicky (Eds.), Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages, QAPL 2005, in: Electronic Notes in Theoretical Computer Science, vol. 153, Elsevier Science Publishers B.V., 2006, pp. 161–175.
[20] R. De Nicola, D. Latella, M. Massink, Formal modeling and quantitative analysis of KLAIM-based mobile systems, in: H. Haddad, L. Liebrock, A. Omicini, R. Wainwright, M. Palakal, M. Wilds, H. Clausen (Eds.), Applied Computing 2005. Proceedings of the 20th Annual ACM Symposium on Applied Computing, Association for Computing Machinery — ACM, ISBN: 1-58113-964-0, 2005, pp. 428–435.
[21] R. De Nicola, M. Loreti, A modal logic for mobile agents, ACM Transactions on Computational Logic 5 (1) (2004) 79–128.
[22] R. De Nicola, M. Loreti, MoMo: A modal logic for reasoning about mobility, in: Proceedings of the Third International Symposium on Formal Methods for Components and Objects, November 2004, Leiden, The Netherlands — FMCO 2004, in: LNCS, vol. 3657, Springer-Verlag, 2005.
[23] R. De Nicola, F. Vaandrager, Action versus state based logics for transition systems, in: I. Guessarian (Ed.), Proceedings of LITP Spring School on Theoretical Computer Science, in: LNCS, vol. 469, Springer-Verlag, 1990, pp. 407–419.
[24] A. Fantechi, S. Gnesi, G. Mazzarini, How much expressive are LOTOS expressions? in: J. Quemada, J. Manas, M. Thomas (Eds.), Formal Description Techniques — III, North-Holland Publishing Company, 1991.
[25] G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, A model-checking verification environment for mobile processes, ACM Transactions on Software Engineering and Methodology 12 (4) (2003) 440–473.
[26] H. Hansson, B. Jonsson, A logic for reasoning about time and reliability, Formal Aspects of Computing. The International Journal of Formal Methods 6 (5) (1994) 512–535.
[27] S. Hart, M. Sharir, Probabilistic temporal logics for finite and bounded models, in: R. De Millo (Ed.), 16th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery — ACM, ISBN: 0-89791-133-4, 1984, pp. 1–13.
[28] H. Hermanns, U. Herzog, J.-P. Katoen, Process algebra for performance evaluation, Theoretical Computer Science 274 (1–2) (2002) 43–87.
[29] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, M. Siegle, Model checking stochastic process algebra, Technical Report IMMD7-2/00, Univ. of Nuernberg, 2000.
[30] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, M. Siegle, Towards model checking stochastic process algebra, in: W. Grieskamp, T. Santen, B. Stoddart (Eds.), Integrated Formal Methods — IFM 2000, in: LNCS, vol. 1945, Springer-Verlag, 2000, pp. 420–439.
[31] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, M. Siegle, A tool for model-checking Markov chains, International Journal on Software Tools for Technology Transfer 4 (2) (2003) 153–172.
[32] J. Hillston, Process algebras for quantitative analysis, in: IEEE Symposium on Logic in Computer Science, IEEE, Computer Society Press, 2005, pp. 239–248.
[33] J.-P. Katoen, M. Khattri, I. Zapreev, A Markov reward model checker, in: Second International Conference on the Quantitative Evaluation of Systems, QEST'05, IEEE Computer Society Press, ISBN: 0-7695-0418-3, 2005, pp. 243–244.
[34] V. Kulkarni, Modeling and Analysis of Stochastic Systems, Chapman & Hall, 1995.
[35] M. Kwiatkowska, G. Norman, D. Parker, Probabilistic symbolic model checking using PRISM: A hybrid approach, International Journal on Software Tools for Technology Transfer 6 (2) (2004) 128–142.
[36] S. Merz, M. Wirsing, J. Zappe, A spatio-temporal logic for the specification and refinement of mobile systems, in: M. Pezzé (Ed.), Fundamental Approaches to Software Engineering, FASE 2003, in: LNCS, vol. 2621, Springer-Verlag, 2003, pp. 87–101.