MoSL: A Stochastic Logic for StoKlaim *

Rocco De Nicola
Università degli Studi di Firenze

Joost-Pieter Katoen
RWTH Aachen

Diego Latella
C.N.R. - I.S.T.I.

Michele Loreti
Università degli Studi di Firenze

Mieke Massink
C.N.R. - I.S.T.I.

ISTI Report version 3.0 of Sep. 14, 2006

Abstract

The programming and modeling language Klaim has been designed to address key functional aspects of global computing such as distribution awareness, (code and agent) mobility, and privacy aspects. This paper concentrates on their integration with performance and dependability aspects, the logical characterization of performance and dependability requirements, and the automatic validation of system models against integrated formal functional and performance/dependability requirements. To that purpose the temporal logic MoSL is introduced for formulating properties of models specified in StoKlaim, a Markovian extension of Klaim introduced in previous work of the authors. MoSL is inspired by (an action-based version of) CSL, an extension of CTL with ample means to refer to performance and dependability aspects. It is shown that a substantial fragment of the logic can be mapped onto the input language of existing probabilistic model checkers, thus allowing for the automated verification of qualitative and quantitative properties of network-aware programs. The approach is illustrated by modeling and verifying the spreading of a virus through a network.

1 Introduction

This paper attempts to make a considerable step into the direction of the development of systematic methods, techniques and tools—all based on solid logic and mathematical foundations i.e., *formal methods*—needed for extablishing performance and dependability requirements and attributes of *dependable global computers*. In particular, we develop an extension of a widely used temporal logic, CTL, as property specification language for distribution, performance and dependability guarantees. The temporal logic formalism builds upon an action-based variant of CSL (Continuous Stochastic Logic [2, 6]).

1.1 Modeling Dependable Global Computers

Global computers [11] are modern, complex distributed systems which are highly dynamic and have to deal with frequent changes of the network environment. Features such as distribution awareness and code mobility which were absent or deliberately invisible in previous computer generations play a prominent rôle in global computing.

*The work presented in this report has been partially supported by EU Project Software Engineering for Service-Oriented Overlay Computers (SENSORIA, contract IST-3-016004-IP-09).

To facilitate the incorporation of random phenomena in models for network-aware computing, we proposed StoKlaim [16], a simple, yet powerful extension of Klaim [14, 8]. Klaim is an experimental language for distributed systems that is aimed at modeling and programming mobile code applications, i.e., applications for which exploiting code mobility is the prime distinctive feature. Its distinguishing feature is the explicit use of localities for modeling data or computational resources distribution. It is heavily based on the process algebras CCS and π-calculus as well as on the coordination paradigm of Linda. Klaim models *networks* as finite collections of sites, each equipped with a (physical) address, where processes can execute and data can reside. Processes and actions—like in traditional process algebras—are key elements in Klaim and possess the possibility to explicitly refer and control the spatial structure of the global network at any point of their evolution. Processes are the active computational entities and may run concurrently, either at the same site or at distinct ones. They interact in an asynchronous fashion via multiple distributed *tuple spaces*, a generalization of the well-known single shared tuple space in Linda [25]. Actions in Klaim explicitly indicate the (possibly remote) site at which they will have effect. Klaim supports core aspects for global computing such as *process distribution*, *remote evaluation* (a process sends another process for execution to another site), *code on demand* (a process may download code from a remote site to execute it locally), and *site creation*.

In StoKlaim, these actions have 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 performance and dependability of information processing systems.

1.2 Specifying properties of dependable global computing

Models specified in StoKlaim thus yield a Markov chain as operational model. To assess dependability aspects, typically long-run or transient probabilities of such chains are determined. 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 in proper temporal logics. For CTMCs, the logic CSL (Continuous Stochastic Logic) [2, 6] is of particular interest for which efficient model-checking algorithms exist. Several software tools support the verification of CSL; this ranges from tailored probabilistic model checkers such as PRISM [31] to tools for stochastic Petri nets [13]. CSL is a stochastic extension of CTL, and allows for—besides the qualitative properties such as safety and liveness—the specification of (time-bounded) probabilistic reachability properties, such as "the likelihood to reach a goal state within t time units while visiting only legal states is at least 0.92".

This paper proposes a stochastic logic that permits to refer to the spatial structure of the network for the specification of properties for StoKlaim models. We take as starting-point an action-based variant of CSL (as first proposed in [28]); this fits well with the action-based nature of Klaim. (The relation between action-based CSL and CSL is similar to that between CTL and action-based CTL [21].) The novel features of this stochastic logic, which we call *Mobile Stochastic Logic* (MoSL), are: atomic propositions may refer to the sites where data and processes reside, actions are generalized to *action specifiers* that act as patterns to characterize sets of actions, and logical variables are incorporated to refer to dynamically created sites. Interestingly enough, a substantial fragment of this logic can directly and efficiently be mapped onto action-based CSL, as shown in this paper. This allows for the use of existing model checkers such as ETMCC [29] for action-based CSL to assess dependability constraints of StoKlaim processes in a fully automated manner. Our approach is exemplified by modeling the spreading of a virus through a network, and by verifying properties such as "the probability that the virus is spread to a specific site within a certain time interval is at most 10^{-4}". A preliminary version of the logic MoSL has been presented in [17].

1.3 Related Work

Several (temporal) logics have been proposed which aim at describing properties of systems related either to mobility ([9, 19, 10, 12, 24, 32] among others) or to probabilistic/stochastic behaviour (e.g. [26, 27, 2, 6, 28]). To the best of our knowledge, the present paper is the first approach towards a probabilistic logic for mobility, except for [17], where a preliminary version of the logic has been proposed which is closely related to the language presented in [18] and suffers of similar restrictions.

1.4 Organization of the paper

Section 2 recalls the formal definition of the modeling language StoKlaim. The property specification language MoSL is introduced in Section 3 together with its formal semantics. Section 4 presents the translation of a major fragment of the logic onto action-based CSL. The correctness proof of this translation is provided in the Appendix. Section 5 recalls the virus spreading example described in [16] and presents a formal analysis of a number of relevant properties using probabilistic model checking. In Section 6 some concluding remarks are drawn.

2 StoKlaim

This section recalls StoKlaim syntax and semantics. A revised version of the formal definition of the language is given which slightely improves that proposed in [16].

2.1 Syntax of StoKlaim

Let $\mathcal{V}$, ranged over by $v, v', v_1, \ldots$, be a set of (basic data) *values*; $\mathcal{I}$, ranged over by $i, i', i_1, \ldots$, be a set of (physical) *addresses*; $\mathcal{L}$, ranged over by $l, l', l_1, \ldots$, be a set of *logical addresses*, also called *localities*; the locality $\mathsf{self} \in \mathcal{L}$; $\mathcal{R}$, ranged over by $r, r', r_1, \ldots$, be a set of *rate names*; $\mathcal{V}$-var, ranged over by $x, x', x_1, \ldots$, be a a set of *value variables*; $\mathcal{L}$-var, ranged over by $u, u', u_1, \ldots$, be a set of *locality variables*; and $\mathcal{P}$-var, ranged over by $X, X', X_1, \ldots, Q, Q', Q_1, \ldots$ be a set of *process variables*. We will conventionally use $Q, Q', Q_1, \ldots$ for those process variables for which there is a proper definition in the StoKlaim specification at hand, as described below.

All these sets are countable and are mutually disjoint. $\mathcal{R}$ is assumed equipped with a decidable equivalence relation $\simeq$ such that $\mathcal{R}/\simeq$ is countable and each element of $\mathcal{R}/\simeq$ is countable as well. Furthermore, let ℓ, ℓ', ℓ_1 range over $\mathcal{L} \cup \mathcal{L}$-var. Finally, we assume 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.

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 element they contain, i.e., $\{s\}$ is denoted as s and $\vec{s} = s'$ as s'. The empty sequence is denoted by ϵ.

The syntax of StoKlaim *nets* is given in Table 1.

2.1.1 Nets and processes

Specifications in StoKlaim consist of nets and processes. 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*. Allocation environment ρ maps localities occurring in the processes running at i to addresses. Node elements are either processes executing at a node—*process nodes* in the sequel—or data (represented as a tuple $\vec{f}$, see later on) that is stored at a node: Processes are built up from the terminated process **nil**, a set of randomly delayed actions, and standard process algebraic constructors such as prefix, choice, parallel composition and process instantiation Q, with optional parameters $(\vec{Q'}, \vec{\ell}, \vec{e})$, where each process variable $Q, Q', \ldots$

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

Table 1: Syntax of StoKlaim.

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:

$$Q(!\vec{X}, !\vec{u}, !\vec{x}) \triangleq P \quad .$$

In the sequel, by *process constant* we mean a process variable for which there is a proper defining equation in the context at hand. For the sake of simplicity, actual process parameters in process instantiations are restricted to be process constants only. Note that, for syntactical uniformity, *all* binding occurrences of variables are prefixed with '!'. This includes occurrences in node creation and read actions (see below), and formal parameters of process definitions.

The process $(A, r).P$ executes action A with a duration that is distributed exponentially with a rate specified by rate-name r. Rate-names are mapped to rate values by means of *rate-mappings*. A rate-mapping β is a *total* function[1] from $\mathcal{R}$ to $\mathbb{R}_{>0}$ such that $\beta\ r = \beta\ r'$ whenever $r \simeq r'$.

Thus, the duration of the execution of action A is exponentially distributed with rate $(\beta\, r)$.

It is convenient to introduce the following notions. Let N be a net. The *site* (with address) i is the collection of nodes in N with address i. Nodes are syntactical 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 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 resp.) occurring as fields of tuples $\vec{f}$ such that $i ::_\rho \langle \vec{f} \rangle$ is in N.

2.1.2 Actions

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 pattern, or *template* $(F_1, \ldots, F_n)$, from repository l. Processes can be written to/withdrawn from a repository as well. In particular, when a process is written to a remote repository, it looses the links of its localities to the addresses they are bound to by the local allocation environment; if and 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 (not shown in Table 1 for the sake of notational simplicity).

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

[1] In this paper we will often use a functional programming like notation where currying will be used in function application, i.e. $f\ a_1\ a_2 \ldots\ a_n$ will be used instead of $f(a_1, a_2, \ldots, a_n)$ and function application will be considered left-associative. We let $\mathsf{dom}\, f$ ($\mathsf{rng}\, f$, respectively) denote the *domain* (*range* respectively) of function f.

dynamic scoping rule is used by default, while the *static* one can be enforced by the asterisk prefix (i.e. $\mathbf{eval}(*P)@l$). A locality variable u can be used in place of l in all above actions. Action $\mathbf{newloc}(!u)$ creates a new node. The newly created locality (bound to the address of the newly created node) is referred to by variable u. Finally, we will use the notation $\mathbf{busy}(r).P$ as an abbreviation 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.

2.1.3 Tuples and templates

Tuple fields can be processes, localities, locality variables and value expressions. For the sake of simplicity, process fields are restricted to process instantiations only, as in the case of process actual parameters of process instantiations. We assume a standard way for building value expressions from values, value variables and operators and do not discuss these in any further detail here. *Template fields* can be tuple fields, or *binders*, which are variables prefixed with an exclamation mark. Binders indicate the binding occurrences of related variables; their *scope* will be defined in Section 2.2.

2.1.4 StoKlaim specifications

A StoKlaim *specification* $\mathcal{S}$ 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 modeling the behaviour of a system and the process definitions for the processes used in N_0.

2.2 Semantics of StoKlaim

In this section the operational semantics of StoKlaim is recalled. Some (straightforward) static semantics constraints are given below.

2.2.1 Well-formed specifications

StoKlaim is a typed language; type-checking StoKlaim is out of the scope of the present paper, where type-correctness of StoKlaim specifications is assumed. Free and bound variables are defined in the usual way: in processes of the form $(\mathbf{newloc}(!u), r).P$, binder $!u$ binds all free occurrences of variable u in P. Similarly, in process $(\mathbf{in}(\vec{F})@\ell, r).P$ or $(\mathbf{read}(\vec{F})@\ell, r).P$ a binder occurring in $\vec{F}$ binds all free occurrences of the variable with the same name in P. In a process definition like $Q(!\vec{Q}', !\vec{u}, !\vec{x}) \triangleq P$, a binder occurring in the formal parameter list $!\vec{Q}', !\vec{u}, !\vec{x}$ binds all free occurrences of the variable with the same name in P. In these cases, P is called the *scope* of the binder at hand.

In the following, for StoKlaim specification $\mathcal{S} = (\beta_0, N_0, \vec{D})$, we let $(\mathsf{Rat}\,\mathcal{S})$, $(\mathsf{Loc}\,\mathcal{S})$, and $(\mathsf{Adr}\,\mathcal{S})$ denote the set of rate names, 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 $(\mathsf{Rat}\,N)$, $(\mathsf{Loc}\,N)$, and $(\mathsf{Adr}\,N)$, for network N as an abbreviation for $(\mathsf{Rat}\,(\beta, N, \epsilon))$, $(\mathsf{Loc}\,(\beta, N, \epsilon))$, and $(\mathsf{Adr}\,(\beta, N, \epsilon))$, for any rate mapping β.

Definition 2.1 A StoKlaim *specification* $(\beta_0, N_0, \vec{D})$ is *well-formed* if and only if it is type-correct and:

- all rate-names occurring in N_0 or $\vec{D}$ are distinct.
- each allocation environment ρ in N_0 satisfies:

  (i) $\mathsf{self} \in (\mathsf{dom}\,\rho)$;

  (ii) $(\mathsf{dom}\,\rho) \setminus \{\mathsf{self}\} \subseteq (\mathsf{Loc}\,N_0)$

  (iii) $(\mathsf{rng}\,\rho) \subseteq (\mathsf{Adr}\,N_0)$

(iv) for all nodes $i_1 ::_{\rho_1} E_1$ and $i_2 ::_{\rho_2} E_2$, if $i_1 = i_2$ then ρ_1 and ρ_2 are *compatible*, i.e., for all l in $(\mathsf{dom}\,\rho_1) \cap (\mathsf{dom}\,\rho_2)$ we have $\rho_1\, l = \rho_2\, l$.

- The only free variables occurring in N_0 are process variables defined in $\vec{D}$.
- All process variables used in the left-hand-side of defining equations in $\vec{D}$ are distinct. In every defining equation $Q(!\vec{Q}', !\vec{u}, !\vec{x}) \triangleq P$ in $\vec{D}$ all binders occurring in the formal parameter list $!\vec{Q}', !\vec{u}, !\vec{x}$ are distinct and all free (process) variables occurring in P, which are not bound by the binders in $!\vec{Q}', !\vec{u}, !\vec{x}$, are defined in some defining equation in $\vec{D}$. Finally, for each process formal parameter $!Q'$ there is at most *one* free occurrence of Q' in P.
- All processes instantiations $Q(\vec{Q}', \vec{\ell}, \vec{e})$ are *guarded*, i.e. they occur in the context of an action prefix $(A, r).Q(\vec{Q}', \vec{\ell}, \vec{e})$, for some A and r.
- In processes of the form $(\mathbf{in}(\vec{F})@\ell, r).P$ or $(\mathbf{read}(\vec{F})@\ell, r).P$, all binders occurring in $\vec{F}$ are distinct; moreover for each process binder $!X$ occurring in $\vec{F}$ there is at most one free occurrence of X in P.

$\diamond$

In the remainder of this paper we assume specifications to be *well-formed*.

2.2.2 Structural congruence

StoKlaim specifications will be mapped by the operational semantics definition onto labeled transition systems, in our case action-labeled Markov chains. The states of these structures are called *configurations*, i.e., tuples (R, I, L, N), often denoted as $R, I, L \vdash N$, where $R \subseteq \mathcal{R}$, $I \subseteq \mathcal{I}$, and $L \subseteq \mathcal{L}$ are the set of rate names, addresses and localities, respectively, in the net N. Configurations are considered modulo the *structural congruence* defined as the least congruence induced by the laws in Table 2. (For the sake of simplicity we have omitted the components R, I, and L when they are unaffected.) Compared to the structural congruence laws of Klaim, the laws (CO+), (AS+), and (NE+), and (REN) have been added. Law (REN) states that any rate name r occurring in N can be replaced by an equivalent rate name, not occurring already in N. The rate name set R must be manipulated accordingly. We adopt the usual notation for syntactical substitution, namely $N[r'/r]$; furthermore, in the rest of this paper, we use the notation $R[r'/r]$ as a shorthand for $(R \cup \{r'\}) \setminus \{r\}$. Finally, in law (CLO) the allocation environments must be taken care of; in particular, ρ_1 and ρ_2 must be *compatible* (see Def. 2.1), in which case, allocation environment $\rho_1 \bowtie \rho_2$ is defined as follows:

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

2.2.3 Tuple evaluation

Function $[\![\cdot]\!]_{\cdot}$ (cf. Table 3) evaluates tuples and templates. Notice that $[\![u]\!]_{\rho}$ yields u. In practice, the static semantics constraints together with the semantics of the **in** and **newloc** actions as well as process instantiation guarantee that variables are properly replaced by their values whenever necessary[2]. In Table 3 function $\mathcal{E}[\![\cdot]\!]$ is used for evaluating value expressions e. The definition of $\mathcal{E}[\![\cdot]\!]$ is outside the scope of the present paper.

$P\{\rho\}$ denotes a process *closure*, i.e., a pair consisting of a process and an allocation environment. $P\{\rho\}$ behaves like process 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. Closures are not part of the language, but are only used in the operational semantics (definition); they may occur at any place where a process is allowed.

[2]Thus, there is no need for explicitly evaluating variables by $[\![\cdot]\!]_{\cdot}$.

(NE$\parallel$)	$N \equiv N \parallel \mathbf{0}$
(CO$\parallel$)	$N_1 \parallel N_2 \equiv N_2 \parallel N_1$
(AS$\parallel$)	$N_1 \parallel (N_2 \parallel N_3) \equiv (N_1 \parallel N_2) \parallel N_3$
(NE+)	$i ::_\rho P \equiv i ::_\rho P + \mathbf{nil}$
(CO+)	$i ::_\rho P_1 + P_2 \equiv i ::_\rho P_2 + P_1$
(AS+)	$i ::_\rho P_1 + (P_2 + P_3) \equiv i ::_\rho (P_1 + P_2) + P_3$
(NE|)	$i ::_\rho P \equiv i ::_\rho P \mid \mathbf{nil}$
(CLO)	$i ::_{\rho_1 \bowtie \rho_2} P_1 \mid P_2 \equiv i ::_{\rho_1} P_1 \parallel i ::_{\rho_2} P_2$ if ρ_1 and ρ_2 are compatible
(REN)	$R \vdash N \equiv (R \cup \{r'\}) \setminus \{r\} \vdash N[r'/r]$ for any $r' \notin (\mathsf{Rat}\, N)$ with $r' \simeq r$

Table 2: Structural congruence laws

$[\![P]\!]_\rho$	$\stackrel{\text{def}}{=} P$	$[\![!X]\!]_\rho$	$\stackrel{\text{def}}{=} !X$
$[\![*P]\!]_\rho$	$\stackrel{\text{def}}{=} P\{\rho\}$	$[\![!u]\!]_\rho$	$\stackrel{\text{def}}{=} !u$
$[\![\ell]\!]_\rho$	$\stackrel{\text{def}}{=} \ell$	$[\![!x]\!]_\rho$	$\stackrel{\text{def}}{=} !x$
$[\![e]\!]_\rho$	$\stackrel{\text{def}}{=} \mathcal{E}[\![e]\!]$		
$[\![(F_1, \ldots, F_n)]\!]_\rho$	$\stackrel{\text{def}}{=} ([\![F_1]\!]_\rho, \ldots, [\![F_n]\!]_\rho)$		

Table 3: Tuple evaluation

$$
\begin{array}{rcl}
ren(\mathbf{nil}, R) & \stackrel{\text{def}}{=} & (\mathbf{nil}, R) \\
ren((A, r).P, R) & \stackrel{\text{def}}{=} & ((A, r').P', R') \text{ where} \\
 & & r' \in \mathcal{R} \setminus R \text{ and} \\
 & & (P', R') = ren(P, R \cup \{r'\}) \\
ren(P_1 \; op \; P_2, R) & \stackrel{\text{def}}{=} & (P_1' \; op \; P_2', R'),\ op \in \{+, |\} \text{ where} \\
 & & (P_1', R'') = ren(P_1, R) \text{ and} \\
 & & (P_2', R') = ren(P_2, R'') \\
ren(Q(\vec{Q}', \vec{\ell}, \vec{e}), R) & \stackrel{\text{def}}{=} & (Q(\vec{Q}', \vec{\ell}, \vec{e}), R) \\
ren(P\{\rho\}, R) & \stackrel{\text{def}}{=} & (P'\{\rho\}, R') \text{ where} \\
 & & (P', R') = ren(P, R) \\
ren(\ell, R) & \stackrel{\text{def}}{=} & (\ell, R) \\
ren(e, R) & \stackrel{\text{def}}{=} & (e, R) \\
ren((f_1, \ldots, f_n), R) & \stackrel{\text{def}}{=} & ((f_1', \ldots, f_n'), R') \text{ where} \\
 & & (f_1', R') = ren(f_1, R) \text{ and} \\
 & & (f_j', R_j') = ren(f_j, R_{j-1}') \\
 & & \text{for } 1 < j \leq n \text{ with } R' = R_n'
\end{array}
$$

Table 4: Rate name renaming

2.2.4 Rate name renaming

Function *ren* (cf. Table 4) takes as argument a process P and a rate name set R. It renames all rate-names occurring in P into fresh names and adapts R accordingly. Function *ren* will be used in defining the semantics of the actions **out** and **eval** and in process instantiation to guarantee unique rate-names. It is not difficult to establish that *ren* indeed generates unique rate names. Notice also that, strictly speaking, Table 4 characterizes a set of functions, each specific one being defined by the particular choices performed in the second equation.

2.2.5 Substitutions and Matching

In the inference rules defined below we exploit substitutions and combinations thereof. They have the usual meaning, i.e., for $d_1, \ldots, d_n$ ranging over $\mathcal{L} \cup \mathcal{V} \cup P$, and $w_1, \ldots, w_n$ ranging over $\mathcal{L}\text{-var} \cup \mathcal{V}\text{-var} \cup \mathcal{P}\text{-var}$, we let $[d_1/w_1 \ldots d_n/w_n]$, with $w_i \neq w_j$ for $i \neq j$, denote the substitution 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:

$$[d_1''/w_1', .., d_m''/w_m', d_{m+1}''/w_{m+1}', .., d_{m+h}''/w_{m+h}']$$

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

$$[d_1/w_1, \ldots, d_n/w_n, d_1''/w_1', \ldots, d_m''/w_m', d_{m+1}''/w_{m+1}', \ldots, d_{m+h}''/w_{m+h}'] \quad .$$

The pattern matching function *match* (cf. Table 5) yields a substitution if a matching is successful. Here, it is assumed that β is the rate mapping for which the matching is considered. (Strictly speaking, β is a parameter of *match*, but as it is unchanged in all cases, this is left implicit for the sake of readability.) In the definition of *match*, processes are considered the same as closures with an empty allocation environment.

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

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

$$\frac{match(P,P') = [] \quad (\beta\, r) = (\beta\, r')}{match((A,r).P,(A,r').P') \stackrel{\text{def}}{=} []} \qquad \frac{match(P_1,P_1') = [] \quad match(P_2,P_2') = []}{match(P_1+P_2,P_1'+P_2') \stackrel{\text{def}}{=} []}$$

$$\frac{match(\vec{P},\vec{P'}) = [] \quad match(\vec{l},\vec{l'}) = [] \quad match(\vec{v},\vec{v'}) = []}{match(X(\vec{P},\vec{l},\vec{v}),X(\vec{P'},\vec{l'},\vec{v'})) \stackrel{\text{def}}{=} []} \qquad \frac{match(P,P') = []}{match(P\{\rho\},P'\{\rho\}) \stackrel{\text{def}}{=} []}$$

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

Table 5: Pattern-matching of tuples against templates

2.2.6 Labeled transition system semantics

Let *RepCnf*, ranged over by $c, c', c_1, \ldots$, be the set of all representatives of the equivalence classes induced by the structural congruence $\equiv$. We abstract here from the way in which such representatives are chosen. We let $rep(R,I,L,N)$ denote the representative of the congruence class of configuration (R,I,L,N).

The *derivatives* $(Der\, c)$ of configuration $c \in RepCnf$ is the smallest set such that it includes c, and is closed under the congruence laws and reduction rules defined in Tables 2 and 6.

The following definition characterizes the LTS associated to a StoKlaim specification. In the definition set $\mathcal{A}$ is the set of *ground* actions constructed according to the grammar below:

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

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

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

Definition 2.2 *The LTS of* StoKlaim *specification* $\mathcal{S} = (\beta_0, N_0, \vec{D})$ *is the tuple* $(C, \Lambda, \longrightarrow, c_0)$ *with:*

- $c_0 \stackrel{def}{=} rep(\mathsf{Rat}\mathcal{S}, \mathsf{Adr}\mathcal{S}, \mathsf{Loc}\mathcal{S}, N_0)$*, is the initial state;*
- $C \stackrel{def}{=} \{rep(c) \mid c \in (Der\, c_0)\}$ *is the set of states;*
- $\longrightarrow$*, the transition relation, is the smallest relation on* $C \times ((\mathcal{I} \times \mathcal{A}) \times \mathcal{R}) \times C$ *induced by the the congruence laws and reduction rules defined in Table 2 and in Table 6;*
- $\Lambda \stackrel{def}{=} \{(\gamma, r) \mid \exists c, c' \in C.\ (c,(\gamma,r),c') \in \longrightarrow\} \subseteq ((\mathcal{I} \times \mathcal{A}) \times \mathcal{R})$ *is the label-set.*

We let $\gamma, \gamma', \gamma_1, \ldots$ range over action-labels and denote $(c,(\gamma,r),c') \in \longrightarrow$ by $c \xrightarrow{\gamma,r} c'$. Finally, we let N_c (R_c, I_c, L_c, respectively) denote the network (rate names, addresses, localities, respectively) component of c.

Given the mapping from StoKlaim specifications onto LTSs, the last step is the mapping of such LTSs onto CTMCs: 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-labeled transition from configuration c to c' should

(OUT) $$\frac{\rho_1\, l = i_2}{R \vdash i_1 ::_{\rho_1} (\mathbf{out}(\vec{f})@l, r).P \parallel i_2 ::_{\rho_2} E \xrightarrow{(i_1, \mathbf{o}(\vec{f''}, i_2)), r} R' \vdash i_1 ::_{\rho_1} P \parallel i_2 ::_{\rho_2} E \parallel i_2 ::_{\rho_2} \langle \vec{f'''} \rangle}$$

where $(\vec{f'}, R') = ren(\vec{f}, R)$ and $\vec{f''} = [\![\vec{f'}]\!]_{\rho_1}$

(IN) $$\frac{\rho_1\, l = i_2}{i_1 ::_{\rho_1} (\mathbf{in}(\vec{F})@l, r).P \parallel i_2 ::_{\rho_2} \langle \vec{f} \rangle \xrightarrow{(i_1, \mathbf{i}(\vec{f}, i_2)), r} i_1 ::_{\rho_1} P\Theta \parallel i_2 ::_{\rho_2} \mathbf{nil}}$$

where $match([\![\vec{F}]\!]_{\rho_1}, \vec{f}) = \Theta$

(RD) $$\frac{\rho_1\, l = i_2}{i_1 ::_{\rho_1} (\mathbf{read}(\vec{F})@l, r).P \parallel i_2 ::_{\rho_2} \langle \vec{f} \rangle \xrightarrow{(i_1, \mathbf{r}(\vec{f}, i_2)), r} i_1 ::_{\rho_1} P\Theta \parallel i_2 ::_{\rho_2} \langle \vec{f} \rangle}$$

where $match([\![\vec{F}]\!]_{\rho_1}, \vec{f}) = \Theta$

(EVL) $$\frac{\rho_1\, l = i_2}{R \vdash i_1 ::_{\rho_1} (\mathbf{eval}(P')@l, r).P \parallel i_2 ::_{\rho_2} E \xrightarrow{(i_1, \mathbf{e}(P'', i_2)), r} R' \vdash i_1 ::_{\rho_1} P \parallel i_2 ::_{\rho_2} E \parallel i_2 ::_{\rho_2} P'''}$$

where $(P'', R') = ren(P', R)$ and $P''' = [\![P'']\!]_{\rho_1}$

(NLC) $$\frac{i_2 \in \mathcal{I} \setminus I \qquad l_2 \in \mathcal{L} \setminus L}{I, L \vdash i_1 ::_{\rho_1} (\mathbf{newloc}(!u), r).P \xrightarrow{(i_1, \mathbf{n}(i_2)), r} I', L' \vdash i_1 ::_{\rho_1'} P[l_2/u] \parallel i_2 ::_{\rho_2} \mathbf{nil}}$$

where $I' = I \cup \{i_2\}$, $L' = L \cup \{l_2\}$, $\rho_1' = \rho_1 \bullet [l_2 \mapsto i_2]$ and $\rho_2 = \rho_1' \bullet [\mathsf{self} \mapsto i_2]$

(CLS) $$\frac{I, L, R \vdash i_1 ::_{\rho_1 \bullet \rho_2} P \parallel N \xrightarrow{\gamma, r} I', L', R' \vdash i_1 ::_{\rho_1' \bullet \rho_2} P' \parallel N'}{I, L, R \vdash i_1 ::_{\rho_1} P\{\rho_2\} \parallel N \xrightarrow{\gamma, r} I', L', R' \vdash i_1 ::_{\rho_1'} P'\{\rho_2\} \parallel N'}$$

(PIN) $$\frac{Q(!\vec{X}, !\vec{u}, !\vec{x}) \triangleq P \qquad I, L, R' \vdash i ::_{\rho} P' \parallel N \xrightarrow{\gamma, r} I', L', R'' \vdash N'}{I, L, R \vdash i ::_{\rho} Q(\vec{Q'}, \vec{l}, \vec{v}) \parallel N \xrightarrow{\gamma, r} I', L', R'' \vdash N'}$$

where $(P', R') = ren(P[\vec{Q'}/\vec{X}, \vec{l}/\vec{u}, \vec{v}/\vec{x}], R)$

(CHO) $$\frac{I, L, R \vdash i ::_{\rho} P \parallel N \xrightarrow{\gamma, r} I', L', R' \vdash N'}{I, L, R \vdash i ::_{\rho} P + P' \parallel N \xrightarrow{\gamma, r} I', L', R' \vdash N'}$$

(PAR) $$\frac{I, L, R \vdash N_1 \xrightarrow{\gamma, r} I', L', R' \vdash N'}{I, L, R \vdash N_1 \parallel N_2 \xrightarrow{\gamma, r} I', L', R' \vdash N' \parallel N_2}$$

(STC) $$\frac{R \vdash N \equiv R[r'/r] \vdash N_1 \qquad R[r'/r], I, L \vdash N_1 \xrightarrow{\gamma, r'} R_2, I', L' \vdash N_2 \qquad R_2 \vdash N_2 \equiv R' \vdash N'}{R, I, L \vdash N \xrightarrow{\gamma, r} R', I', L' \vdash N'}$$

Table 6: Reduction rules for StoKlaim

be obtained with rate $(\beta\, r) + (\beta\, r')$. In practice we map LTSs to*action-labeled* CTMCs (AMCs), defined below:

Definition 2.3 *An* action-labeled *CTMC (AMC)* $\mathcal{M}$ *is a triple* $(S, ACT, \mapsto)$ *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* $S \times ACT \times S$ *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.

Assume that the LTS associated to the StoKlaim specification is finite, i.e., it is finitely branching and has a finite number of states[3].The following definition characterizes the AMC associated to a StoKlaim specification.

Definition 2.4 *For* StoKlaim *specification* $(\beta_0, N_0, \vec{D})$ *with finite LTS* $(C, \Lambda, \longrightarrow, c_0)$*, let* $\mathsf{AMC}(\beta_0, N_0, \vec{D}) \stackrel{def}{=} (S, ACT, \mapsto)$ *with:*

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

3 MoSL

The previous section has introduced StoKlaim and its semantics that results in a Markov chain. This section presents a logic that, basically, has the following features:

- it is a *temporal logic* that permits describing the dynamic evolution of the system
- it is a *real-time logic* that permits describing rela-time bounds in the dynamic evolution of the system
- it is a *probabilistic logic* that permits expressing not only functional properties, but, in particular, also properties related to performance and dependability aspects, and, finally
- it is a *spatial logic* that permits referring to the spatial structure of the network for the specification.

We start by presenting the syntax and semantics of the logic, that we called MoSL (which abbreviates Mobile CSL), and then consider the more practical issue of mapping a substantial fragment of this logic onto action-based CSL [28] for which efficient model checkers exist.

3.1 Syntax

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

[3]There are several ways for assuring finiteness of transition systems obtained from process algebras; see, e.g., [23]. We will not dwell further upon this issue here.

3.1.1 Atomic propositions

The most elementary formulae that can be specified are atomic propositions. They come in two shapes. Atomic proposition $Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath$ holds whenever process $Q(\vec{Q}', \vec{\ell}, \vec{e})$ is running at site $\imath$. Atomic proposition $\langle \vec{F} \rangle @\imath$ holds whenever a tuple $\vec{f}$ is stored at site $\imath$ and matches $\vec{F}$. The grammar for propositions is thus:

$$\aleph \quad ::= \quad P@\imath \mid \langle \vec{F} \rangle @\imath \quad .$$

Recall that $\imath$ is either a physical address or an address variable. These variables are assigned values by means of action specifiers (see below). Localities are—in contrast to the modeling language—*not* used for identifying sites in the logic. This stems from the fact that localities have more a local connotation (which is resolved by allocation environments), while at the property specification level one has a global view of the entire network.

3.1.2 Action specifiers and action sets

As in the branching-time temporal logic CTL, formulae in our logic can either refer to a state or to a path. As we deal with an 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 [21]. 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:

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

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

$$\xi ::= g : \mathbf{o}(\vec{F}, g) \mid g : \mathbf{i}(\vec{F}, g) \mid g : \mathbf{r}(\vec{F}, g) \mid g : \mathbf{e}(F, g) \mid g : \mathbf{n}(g)$$

where g is an address template, i.e., g is either of the form ι or $!z$. Action specifier $i_1 : \mathbf{o}(v, i_2)$, say, is satisfied only by action $(i_1, \mathbf{o}(v, i_2))$. The occurrence of this action models the uploading of value v at site i_2 by a process at site i_1. Action specifiers may contain binders that bind their variables to corresponding values in actions in the path; e.g., the action specifier $!z_1 : \mathbf{o}(v, !z_2)$ is satisfied by any action, executed at some site, which uploads value v at some site. This action specifier is, e.g., satisfied by action $(i_1, \mathbf{o}(v, i_2))$. The meaning of the other action specifiers is self-explanatory.

3.1.3 Path formulae

A path in an AMC $(S, ACT, \mapsto)$ is an alternating sequence (finite or infinite) of states and pairs consisting of an action and a positive real number. An infinite path is a sequence $s_0\,(\gamma_0, t_0)\,s_1\,(\gamma_1, t_1)\ldots$ such that, for all $j \geq 0$, s_j is a state, t_j is a positive real number, γ_j is an action, and $s_j \overset{\gamma_j, \lambda}{\longmapsto} s_{j+1}$ for some $\lambda > 0$. Intuitively, the sequence traverses the states s_0, s_1, s_2 and so on while staying t_i time units in state s_i and performing γ_i while moving from s_i to s_{i+1}.

The basic format of a path formula is the CTL-formula $\Phi \ \mathcal{U} \ \Psi$. In order to be able to refer to actions executed along a path, we in fact use the variant of until as originally proposed in action-based CTL [21]. 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 on 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 additionally that a Ψ-state should be reached within t time units. If $t = \infty$, this time constraint is vacuously true, and the until from action-based CTL is obtained. Accordingly, the syntax of path formulae is:

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

Note that the only difference between the two until-operators is the absence or presence of the right subscript, i.e., the action set specifying the constraints on the action which must be executed for entering the Ψ-state. We emphasize that $\Phi \ {}_{\Delta}\mathcal{U}^{<t} \Psi$ is *not* equivalent to $\Phi \ {}_{\Delta}\mathcal{U}_{\top}^{\leq t} \Psi$. The precise difference between the two until-formulae will become apparent when defining the semantics (cf. next subsection).

Action specifiers and their matching to actions generate substitutions in a natural way. Consequently, variables may occur in formulae and are replaced by the associated values via substitutions. For example, $\mathtt{tt} \ {}_{\top}\mathcal{U}_{i_1:\mathbf{N}(!z)}$ **nil**@z states that when a new node (referred as) z is created from site i_1, the **nil** process will be "running" on z.

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 the ones introduced before. 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 a path. 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, exceeds 0.99. Here, both the actions to take to move between legal states and the one for entering the goal state are irrelevant, as indicated by the action set $\top$. Long-run properties refer to the system when it has reached an equilibrium. Under the assumption that the CTMC is finite, such equilibrium will always exist [30]. A state s satisfies $\mathcal{S}_{\bowtie p}(\Phi)$ if the probability of reaching from s, in the long run, a state which satisfies Φ is $\bowtie p$. Thus, state-formulae are built according to the grammar:

$$\begin{array}{lcll} \Phi & ::= & \aleph & \text{(atomic proposition)} \\ & \mid & \neg \, \Phi & \text{(negation)} \\ & \mid & \Phi \vee \Phi & \text{(disjunction)} \\ & \mid & \mathcal{P}_{\bowtie p}(\varphi) & \text{(probabilistic path operator)} \\ & \mid & \mathcal{S}_{\bowtie p}(\Phi) & \text{(long-run operator)} \end{array}$$

3.2 Semantics

Paths play a central role in the formal definition of the semantics of MoSL. Let $\mathcal{M} = (S, ACT, \mapsto)$ be an action-labelled CTMC. A *path* π of $\mathcal{M}$ is a sequence

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

such that the following two conditions hold:

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

Path operators which will be used in the sequel are defined in Table 7. For any state s of an AMC $\mathcal{M}$, let $\mathit{Paths}(s)$ denote the set of *all* paths $s_0(\gamma_0, t_0)s_1(\gamma_1, t_1)\ldots$ over $\mathcal{M}$ with $s_0 = s$. A Borel space can be defined over $\mathit{Paths}(s)$, together with its associated probability measure $\mathbb{P}$, which is a slight extension of that defined in [6] in order to take both states and actions into consideration [15].

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

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

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

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

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

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

Table 7: Operators on paths

3.2.1 Well-formed formulae

We will require top-level (state) formulae to be *well-formed* with respect to the net specification $(\beta_0, N_0, \vec{D})$, according to the definition below. The notion of free and bound variable in a logical formula is used in the following sense. Since an action set may consist of more than one action specifier and as different action specifiers may introduce different binders, we define the *binding set* of an action set Δ as the *smallest* set $(\mathsf{BS}\,\Delta)$ of binders involved in the action set; cf. Table 8. In all path formulae of the form $\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}^{<t}\ \Psi$ a binder in $(\mathsf{BS}\,\Omega)$ binds all free occurrences of the variable with the same name in Ψ, which is the *scope* of the binder at hand. Notice that the binders in Δ have an empty scope. The reason for that is that any path π such that $(\mathsf{st}(\pi, 0)$ satisfies Φ and) $\mathsf{ac}(\pi, 0)$ satisfies Ω and brings (within a time less than t) to a Ψ-state $\mathsf{st}(\pi, 1)$ does satisfy $\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}^{<t}\ \Psi$; notice that, in this case, no action specifier in Δ is used, so that variables potentially bound by any such specifier would actually not be bound if used in Φ or Ψ. Similar arguments justify the definition of BS and of the scope of binders in Ω. In conclusion, the only way for binding a variable is by means of Ω and the variable can be used only in Ψ. As we shall see later, the aCTL-like *next* operator $\mathbf{X}_{\Omega}\,\Psi$ can be expressed as $\mathsf{tt}\ {}_{\top}\mathcal{U}_{\Omega}\,\Psi$. So, one can think of MoSL binding operators as those which take effect at the transition matching the action specifiers of the next operator and, consequently, whose substitutions are available in the argument formula of such operator.

MoSL is a many-sorted logic and in particular it contains variables (and related values) of different types; type-checking MoSL is out of the scope of the present paper, where type-correctness is assumed.

Definition 3.1 A top-level MoSL state formula Φ for StoKlaim net specification $(\beta_0, N_0, \vec{D})$ is *well-formed* if it is type-correct and all the following conditions hold:

- No variable occurs free in Φ, except process variables for which proper defining equations are given in $\vec{D}$ (i.e. process constants).
- In any subformula of Φ, of the form $\langle \vec{F} \rangle @ \iota$, all binders occurring in template $\vec{F}$ are distinct.
- In all action specifiers of the form $g_1 : \mathbf{o}(\vec{F}, g_2)$, or $g_1 : \mathbf{i}(\vec{F}, g_2)$, or $g_1 : \mathbf{r}(\vec{F}, g_2)$, or $g_1 : \mathbf{e}(F, g_2)$, or $g_1 : \mathbf{n}(g_2)$, occurring in any sub-formula of Φ, all binders in g_1, $\vec{F}$, and g_2 are

$\mathsf{BS}\ \top$	$\stackrel{\text{def}}{=}$	$\varnothing$
$\mathsf{BS}\ \{\xi_1, \ldots, \xi_n\}$	$\stackrel{\text{def}}{=}$	$\bigcap_{j=1}^{n} (\mathsf{BS}\ \xi_j)$
$\mathsf{BS}\ g_1 : \mathbf{O}(\vec{F}, g_2)$	$\stackrel{\text{def}}{=}$	$(\mathsf{BS}\ g_1) \cup (\mathsf{BS}\ \vec{F}) \cup (\mathsf{BS}\ g_2)$
$\mathsf{BS}\ g_1 : \mathbf{I}(\vec{F}, g_2)$	$\stackrel{\text{def}}{=}$	$(\mathsf{BS}\ g_1) \cup (\mathsf{BS}\ \vec{F}) \cup (\mathsf{BS}\ g_2)$
$\mathsf{BS}\ g_1 : \mathbf{R}(\vec{F}, g_2)$	$\stackrel{\text{def}}{=}$	$(\mathsf{BS}\ g_1) \cup (\mathsf{BS}\ \vec{F}) \cup (\mathsf{BS}\ g_2)$
$\mathsf{BS}\ g_1 : \mathbf{E}(F, g_2)$	$\stackrel{\text{def}}{=}$	$(\mathsf{BS}\ g_1) \cup (\mathsf{BS}\ F) \cup (\mathsf{BS}\ g_2)$
$\mathsf{BS}\ g_1 : \mathbf{N}(g_2)$	$\stackrel{\text{def}}{=}$	$(\mathsf{BS}\ g_1) \cup (\mathsf{BS}\ g_2)$
$\mathsf{BS}\ (F_1, \ldots F_n)$	$\stackrel{\text{def}}{=}$	$\bigcup_{j=1}^{n} (\mathsf{BS}\ F_j)$
$\mathsf{BS}\ F$	$\stackrel{\text{def}}{=}$	$\{u \mid F =!u\} \cup \{x \mid F =!x\} \cup \{X \mid F =!X\}$
$\mathsf{BS}\ g$	$\stackrel{\text{def}}{=}$	$\{z \mid g =!z\}$

Table 8: The binding set of an action set

distinct.

$\diamond$

Example. *A formula like* $\mathsf{tt}\ {}_{i_1:\mathbf{N}(!z)}\mathcal{U}^{<35}_{i_1:\mathbf{O}(!x,i_2)}\ \langle x\rangle @ i_2$ *is well-formed; on the other hand, the formula* $\mathsf{tt}\ {}_{i_1:\mathbf{N}(!z)}\mathcal{U}^{<35}_{i_1:\mathbf{O}(!x,i_2)}\ \langle x\rangle @ z$ *is not, due to the last (free!) occurrence of* z. ■

3.2.2 Satisfaction Relation

In this section the formal definition of the satisfaction relation of MoSL is given with reference to StoKlaim specifications.

Definition 3.2 *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* $s_0 \models \Phi$*, where* s_0 *is the state of* $\mathsf{AMC}(\beta_0, N_0, \vec{D})$ *corresponding to the initial state* c_0 *of the LTS of* $(\beta_0, N_0, \vec{D})$*, as defined in Def. 2.4, and* $\models$ *is defined in Table 10.*

State formulae
The satisfaction relation for state-formulae exploits pattern-matching. To that end, the definition of function *match* given in Table 5 is extended as in Table 9, in order to cover addresses, address variables, and action specifiers (with related actions).

Table 10 gives the definition of the satisfaction relation for MoSL formulae. For deciding if 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. $\pi(t) \models \Phi$) must be computed and it must be checked whether it is $\bowtie p$. State s satisfies $\mathcal{P}_{\bowtie p}(\varphi)$ if the probability mass of the set of paths which satisfy φ is $\bowtie p$. The definition of the satisfaction relation for the other kinds of state formulae is straightforward.

Sets of action specifiers
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. The matching function

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

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

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

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

$$\frac{match(g_1,i_1)=\Theta_1 \quad match(F,P)=\Theta_2 \quad match(g_2,i_2)=\Theta_3}{match(g_1:\mathbf{e}(F,g_2),(i_1,\mathbf{e}(P,i_2))) \stackrel{\text{def}}{=} \Theta_1 \lhd \Theta_2 \lhd \Theta_3}$$

$$\frac{match(g_1,i_1)=\Theta_1 \quad match(g_2,i_2)=\Theta_2}{match(g_1:\mathbf{n}(g_2),(i_1,\mathbf{n}(i_2))) \stackrel{\text{def}}{=} \Theta_1 \lhd \Theta_2}$$

Table 9: Matching of address (variables) and action specifiers

requires the rate-binding β_0 and generates a substitution; moreover, specifier *sets* are used in the formulae. Consequently, the satisfiability relation is defined over *(action, substitution)*-pairs and specifier *sets*; such a set is satisfied if one of its elements is satisfied.

Path formulae
The definition of the satisfiability relation for path formulae formalizes the meaning of the until operators, as discussed in Sect. 3.1. 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. 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 discarded, indeed. Notice that, also in this case, the use of binders does make sense since they can be used as *don't care* placeholders.

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 11, 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.

3.3 On the logical characterization of performance/dependability attributes

We close this section with a brief discussion on how some relevant dependability-related properties involving resource distribution of global overlay computers can be expressed in MoSL. The

$$
\begin{array}{lll}
s \models \mathsf{tt} & & \\
s \models \neg\,\Phi & \text{iff } s \models \Phi \text{ does not hold} & \\
s \models \Phi \vee \Psi & \text{iff } s \models \Phi \text{ or } s \models \Psi & \\
s \models \mathcal{S}_{\bowtie p}(\Phi) & \text{iff } \lim_{t\to\infty} \mathbb{P}\{\pi \in \mathit{Paths}(s) \mid \pi(t) \models \Phi\} \bowtie p & \\
s \models \mathcal{P}_{\bowtie p}(\varphi) & \text{iff } \mathbb{P}\{\pi \in \mathit{Paths}(s) \mid \pi \models \varphi\} \bowtie p & \\
s \models Q(\vec{Q}', \vec{\ell}, \vec{e})@i & \text{iff there exist } N \text{ and } \rho \text{ s.t. } N_s \equiv N \parallel i ::_\rho Q(\vec{Q}', \vec{\ell}, \vec{e}) & \\
s \models \langle \vec{F} \rangle @i & \text{iff there exist } N,\ \rho,\ \vec{f}, \text{ and } \Theta \text{ s.t} & \\
 & \quad N_s \equiv N \parallel i ::_\rho \langle \vec{f} \rangle \text{ and } \mathit{match}(\vec{F}, \vec{f}, \beta_0) = \Theta & \\
 & & \\
\gamma, \Theta \models \top & & \\
\gamma, \Theta \models \{\xi_1, \ldots, \xi_n\} & \text{iff there exists } j,\ 0 < j \leq n, \text{ s.t. } \gamma, \Theta \models \xi_j & \\
\gamma, \Theta \models \xi_j & \text{iff } \mathit{match}(\xi_j, \gamma, \beta_0) = \Theta & \\
 & & \\
\pi \models \Phi\ {}_\Delta\mathcal{U}_\Omega^{<t}\ \Psi & \text{iff there exists } k,\ 0 < k \leq (\mathsf{len}\,\pi) \text{ s.t. the following } \textit{three} \text{ conditions hold:} & \\
 & \quad 1) \quad t > \textstyle\sum_{j=0}^{k-1} \mathsf{dl}(\pi, j) & \\
 & \quad 2) \quad \text{there exists } \Theta_{k-1} \text{ s.t. the following } \textit{three} \text{ conditions hold:} & \\
 & \qquad 2.1)\ \mathsf{st}(\pi, k-1) \models \Phi & \\
 & \qquad 2.2)\ \mathsf{ac}(\pi, k-1), \Theta_{k-1} \models \Omega & \\
 & \qquad 2.3)\ \mathsf{st}(\pi, k) \models \Psi\Theta_{k-1} & \\
 & \quad 3) \quad \text{if } k > 1 \text{ then there exist } \Theta_0, \ldots, \Theta_{k-2} \text{ s.t.} & \\
 & \qquad \text{for all } j,\ 0 \leq j \leq k-2 \text{ the following } \textit{two} \text{ conditions hold:} & \\
 & \qquad 3.1)\ \mathsf{st}(\pi, j) \models \Phi & \\
 & \qquad 3.2)\ \mathsf{ac}(\pi, j), \Theta_j \models \Delta & \\
\pi \models \Phi\ {}_\Delta\mathcal{U}^{<t}\ \Psi & \text{iff } \mathsf{st}(\pi, 0) \models \Psi \text{ or} & \\
 & \quad \text{there exists } k,\ 0 < k \leq (\mathsf{len}\,\pi) \text{ s.t. the following } \textit{three} \text{ conditions hold:} & \\
 & \quad 1) \quad t > \textstyle\sum_{j=0}^{k-1} \mathsf{dl}(\pi, j) & \\
 & \quad 2) \quad \mathsf{st}(\pi, k) \models \Psi & \\
 & \quad 3) \quad \text{there exist } \Theta_0, \ldots, \Theta_{k-1} \text{ s.t.} & \\
 & \qquad \text{for all } j,\ 0 \leq j \leq k-1 \text{ the following } \textit{two} \text{ conditions hold:} & \\
 & \qquad 3.1)\ \mathsf{st}(\pi, j) \models \Phi & \\
 & \qquad 3.2)\ \mathsf{ac}(\pi, j), \Theta_j \models \Delta &
\end{array}
$$

Table 10: Satisfaction relation

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

Table 11: Derived operators

general issue of stochastic/reward temporal logic characterization of performance, dependability and performability features has been addressed in, e.g. [3, 4]; we follow a similar approach here.

Steady-state measures can directly be modeled by means of the $\mathcal{S}()$ operator. For instance, the formula

$$\mathcal{S}_{>0.9}(\langle f \rangle @ i)$$

states that in the long term the probability of finding value f stored at site i is larger than 0.9. For example, f could be a malicious (or faulty) process Q and the formula would then characterize the average fraction of time site i is infected (or contains a faulty component). Similarly the formula

$$\mathcal{S}_{>0.9}(Q @ i)$$

would model the average fraction of time the infection (or faulty component) is active in site i. Proper combinations of state properties can be used for identifying interesting state properties involving more than one site.

Suppose now that retrieval of a specific corrupted value v from site i by a site is known to produce an error which may result in a fault of the receiving site. It might be of interest to know an upper bound, say 0.2, of the probability that such a retrieval is performed within a certain amount of time t. The following *transient* probability formula can be used to the above purpose:

$$\mathcal{P}_{<0.2}({}_{\top}\Diamond^{<t}_{!z:\mathbf{I}(v,i)}\ \mathsf{tt}).$$

It is worth noting here that there are versions of CSL (see e.g. [3, 4]) where more general intervals can be used for expressing time constraints in the until operator. We can use similar generalizations for MoSL as well and express also instantaneous transient probability like in the following version of the above formula (we remind the reader that a substantial fragment of MoSL can be translated into aCSL, as we shall see in the next section):

$$\mathcal{P}_{<0.2}({}_{\top}\Diamond^{[t,t]}_{!z:\mathbf{I}(v,i)}\ \mathsf{tt}).$$

which expresses the fact that the transient probability to enter *at time* t a state by retrieving v from i is smaller than 0.2.

More in general, if the presence of a certain value v stored in site i is a symptom of the site being faulty, the following formula can be used for getting information on the distribution of time to failure:

$$\mathcal{P}_{\bowtie p}(\neg(\langle v \rangle @ i)\ {}_{\top}\mathcal{U}^{[t,t]}\ \langle v \rangle @ i).$$

The above formula can be enriched in order to study the distribution for those failures for which there is an immediate activation of a recovery process Q in the affected site which has a probability greater than 0.85 to perform complete recovery within 5 time units:

$$\mathcal{P}_{\bowtie p}(\neg(\langle v \rangle @ i)\ {}_{\top}\mathcal{U}^{[t,t]}\ (\langle v \rangle @ i \wedge \mathcal{P}_{\geq 1}(\mathbf{X}_{!z:\mathbf{E}(Q,i)}\ \mathcal{P}_{>0.85}({}_{\top}\Diamond^{<5}\ \neg(\langle v \rangle @ i))))).$$

The above is an example of *nested* measures. Another example is the following formula which expresses that, in equilibrium, the probability is at least 0.87 that the system will recover from a fault at site i within 5 time units with probability at least 0.75

$$\mathcal{S}_{\geq 0.87}(\mathcal{P}_{\geq 0.75}(\langle v \rangle @ i\ {}_{\top}\mathcal{U}^{<5}\ \neg(\langle v \rangle @ i))).$$

4 From MoSL to aCSL

In this section we present a translation from a large fragment of MoSL to aCSL and we show its correctness. The fragment includes all MoSL ground formulae, i.e. those where no binders or variables occur. Our conjecture is that the excluded formulae can be represented as proper disjunctions indexed with all possible values the binders can take for a given AMC, as we shall

briefly discuss at the end of the present section. In the following, we let MoSL$^-$ denote the restricted language.

Given a MoSL$^-$ formula Φ, and a StoKlaim specification $(\beta_0, N_0, \vec{D})$, and assuming LTS $\mathsf{LTS}(\beta_0, N_0, \vec{D}) = (C, \Lambda, \longrightarrow, c_0)$ *finite*, with $\mathsf{AMC}(\beta_0, N_0, \vec{D}) = (S, ACT, \mapsto)$ being the related AMC, the question is how to translate $\mathsf{AMC}(\beta, N, \vec{D})$ and Φ into an AMC and an aCSL formula in order to perform model checking using an existing aCSL model-checker.

Since in aCSL only *action* atomic propositions can be expressed—which are directly related to transition labels—the first step of our procedure is concerned with finding a way for incorporating information related to *state* atomic propositions into the transition labels of the original AMC. To that purpose, let $\{\aleph_1, \ldots, \aleph_n\}$ be the set of *all* the atomic propositions occurring in Φ. For notational convenience, we associate a unique name p_j to each $\aleph_j$ above. Our first objective is to associate a unique label to each state s of the AMC encoding which atomic propositions the state satisfies; such a label is a string $y_1 \ldots y_n$ where y_j is equal to p_j if s satisfies $\aleph_j$ and is equal to $\bar{p_j}$ otherwise. Formally, let $B(p_1, \ldots, p_n) \stackrel{\text{def}}{=} \times_{j=1}^{n} \{p_j, \bar{p_j}\}$; we define the *characteristic function* of S as follows:

Definition 4.1 The characteristic function of set S of AMC states is the total function $\chi : S \to B(p_1, \ldots, p_n)$ with $\chi\, s \stackrel{\text{def}}{=} y_1 \ldots y_n$ such that, for $j = 1, \ldots, n$, $y_j = p_j$ if $s \models \aleph_j$ and $y_j = \bar{p_j}$ if $s \models \aleph_j$ does not hold. $\diamond$

Example. If the only atomic propositions occurring in a given formula $\tilde{\Phi}$ are $\aleph_1$ (represented by p_1) and $\aleph_2$ (represented by p_2) and a given state $\tilde{s}$ satisfies $\aleph_2$ but does not satisfy $\aleph_1$, we have $(\chi\, \tilde{s}) = \bar{p_1} p_2 \in B(p_1, p_2) = \{\bar{p_1}\bar{p_2}, \bar{p_1}p_2, p_1\bar{p_2}, p_1 p_2\}$. ■

Notice that the above satisfiability check is computed by a simple analysis of (the network component of) the states.

We transform $\mathsf{AMC}(\beta, N, \vec{D})$ into another AMC, $\mathsf{FAMC}(\beta, N, \vec{D})$, by moving the relevant state information *forward* to the transitions emanating from states.

Definition 4.2 Given AMC $\mathsf{AMC}(\beta_0, N_0, \vec{D}) = (S, ACT, \mapsto)$ and bijective encoding cod of $ACT \times B(p_1, \ldots, p_n)$ into finite set Υ, we define $\mathsf{FAMC}(\beta_0, N_0, \vec{D})$ as the AMC $(S_F, ACT_F, \mapsto_F)$ where $S_F = S$, $ACT_F \subseteq \Upsilon$, and $\mapsto_F$ is such that $s \stackrel{\mathsf{cod}(\gamma, (\chi\, s)), \lambda}{\mapsto_F} s'$ if and only if $s \stackrel{\gamma, \lambda}{\mapsto} s'$. $\diamond$

For every path $\pi \in Paths(s)$ over $\mathsf{AMC}(\beta_0, N_0, \vec{D})$ there is a path π_F over $\mathsf{FAMC}(\beta_0, N_0, \vec{D})$ which corresponds to π in the obvious way: for all $j \geq 0$, $\mathsf{st}(\pi_F, j) = \mathsf{st}(\pi, j)$, $\mathsf{ac}(\pi_F, j) = \mathsf{cod}(\mathsf{ac}(\pi, j), (\chi\, \mathsf{st}(\pi, j)))$, and $\mathsf{dl}(\pi_F, j) = \mathsf{dl}(\pi, j)$. We let $Paths_F(s)$ denote the set of all such paths. It is worth pointing out that this AMC transformation cannot deal properly with *absorbing* states. In the following we assume that $\mathsf{AMC}(\beta_0, N_0, \vec{D})$ does not contain absorbing states (they can be eliminated by equipping them with proper self-loops [5]).

In the remainder of this section, for the sake of conciseness, we will use the convention that $\mathcal{M}$ stands for $\mathsf{AMC}(\beta_0, N_0, \vec{D}) = (S, ACT, \mapsto)$; similarly, $\mathcal{M}_F$ stands for $\mathsf{FAMC}(\beta_0, N_0, \vec{D}) = (S_F, ACT_F, \mapsto_F)$.

The intuition behind the design of our translation is that we take $B(p_1, \ldots, p_n)$ as the domain for an interpretation function R over MoSL$^-$ in such a way that each (boolean combination of atomic) proposition(s) Ψ is mapped into the subset of $B(p_1, \ldots, p_n)$ containing only the disjuncts of the sum-product-form expression representing Ψ, in the usual way.

Example. With reference to our previous example, the sum-product-form representing p_2 in $B(p_1, p_2)$ is $p_1 p_2 + \bar{p_1} p_2$; thus, with reference to our sample formula $\tilde{\Phi}$, we have $(\mathsf{R}\aleph_2) = \{p_1 p_2, \bar{p_1} p_2\}$ and clearly $(\chi\, \tilde{s}) \in (\mathsf{R}\aleph_2)$. ■

This last example gives the hint that satisfiability of a state atomic proposition can be reduced to checking if the characteristic function of the state gives a label which is included in the set

$$
\begin{array}{lcl}
\mathsf{R}\,\mathsf{tt} & \stackrel{\text{def}}{=} & B(p_1,\ldots,p_n) \\
\mathsf{R}\,\aleph_j & \stackrel{\text{def}}{=} & \{y_1\ldots y_n \mid y_j = p_j, y_i \in \{p_i,\bar{p_i}\}, \text{for } j \neq i = 1,\ldots,n\},\ \text{for } j = 1,\ldots,n \\
\mathsf{R}\,(\neg\Phi) & \stackrel{\text{def}}{=} & (\mathsf{R}\,\mathsf{tt}) \setminus (\mathsf{R}\,\Phi),\ \text{if}\,\mathsf{APC}\,(\neg\Phi) \\
 & \stackrel{\text{def}}{=} & (\mathsf{R}\,\mathsf{tt}),\ \text{otherwise} \\
\mathsf{R}(\Phi \vee \Phi') & \stackrel{\text{def}}{=} & (\mathsf{R}\,\Phi) \cup (\mathsf{R}\,\Phi'),\ \text{if}\,\mathsf{APC}(\Phi \vee \Phi') \\
 & \stackrel{\text{def}}{=} & (\mathsf{R}\,\mathsf{tt}),\ \text{otherwise} \\
\mathsf{R}(\mathcal{S}_{\bowtie p}(\Phi)) & \stackrel{\text{def}}{=} & (\mathsf{R}\,\mathsf{tt}) \\
\mathsf{R}(\mathcal{P}_{\bowtie p}(\varphi)) & \stackrel{\text{def}}{=} & (\mathsf{R}\,\mathsf{tt})
\end{array}
$$

where

$$
\begin{array}{lcllcl}
\mathsf{APC}\,\mathsf{tt} & \stackrel{\text{def}}{=} & \mathsf{tt} & \mathsf{APC}(\Phi \vee \Phi') & \stackrel{\text{def}}{=} & (\mathsf{APC}\,\Phi) \wedge (\mathsf{APC}\,\Phi') \\
\mathsf{APC}\,\aleph_j & \stackrel{\text{def}}{=} & \mathsf{tt},\ \text{for } j = 1,\ldots,n & \mathsf{APC}(\mathcal{S}_{\bowtie p}(\Phi)) & \stackrel{\text{def}}{=} & \mathsf{ff} \\
\mathsf{APC}(\neg\,\Phi) & \stackrel{\text{def}}{=} & (\mathsf{APC}\,\Phi) & \mathsf{APC}(\mathcal{P}_{\bowtie p}(\varphi)) & \stackrel{\text{def}}{=} & \mathsf{ff}
\end{array}
$$

Table 12: Function R

associated to the atomic proposition by R. Thus we are moving toward the procedure the aCSL model-checker uses for *action* labels. Of course, we have to integrate the information related to state-atomic propositions, namely the value given by the state-characteristic function, with action-related information of the original AMC and to incorporate the result in the transition labels of the resulting AMC; this is done by means of the bijective encoding cod.

We now complete the characterization of R, defining its behaviour on formulas of MoSL$^-$ other than state-atomic propositions, and show how the result of R is integrated, by means of another function A, with the action specifiers occurring (in the indexes of until operators) in the input formula Φ. The actual translation function, T, will use function A for generating the aCSL formula associated to Φ.

Function R is defined in Table 12 and for each state atomic proposition—or boolean combinations of atomic propositions—it generates the associated representation in $B(p_1,\ldots,p_n)$.

Example. *In our running example we have* $\mathsf{R}\aleph_1 = \{p_1\bar{p_2}, p_1p_2\}$, $\mathsf{R}(\neg\mathsf{tt}) = \varnothing$, *and* $\mathsf{R}\,\mathsf{tt} = \{\bar{p_1}\bar{p_2}, \bar{p_1}p_2, p_1\bar{p_2}, p_1p_2\}$. ■

On formulae containing also stochastic or temporal operators R behaves the same as for tt. Predicate APC characterizes the subset of MoSL$^-$ formulae which do *not* contain modal operators, i.e. formulae which are only state-atomic propositions or boolean combinations of state-atomic propositions. Function R enjoys the property stated by Lemma 4.3 below[4].

Lemma 4.3 *For all* $\mathcal{M}$, *states* s *of* $\mathcal{M}$, *and* MoSL$^-$ *formulae* Φ, *the following holds:*

i) if $\mathsf{APC}(\Phi)$ *then:* $\mathcal{M}, s \models \Phi$ *iff* $(\chi\, s) \in \mathsf{R}(\Phi)$

ii) $\mathcal{M}, s \models \Phi$ *implies* $(\chi\, s) \in \mathsf{R}(\Phi)$ □

Each action specifier ξ of MoSL$^-$ uniquely corresponds to an action $\gamma \in ACT$ in the obvious way. For instance the direct correspondent of action specifier $i_1 : \mathbf{o}((l_1, l_2, v), i_2)$ is $(i_1, \mathbf{o}((l_1, l_2, v), i_2)$. The correspondence is trivially lifted to sets of action specifiers. We let $(\Gamma\,\Delta)$ denote the subset of ACT corresponding to the action set Δ. Moreover we define $(\Gamma\,\top)$ as the set of all $\gamma \in ACT$ occurring in transitions of $\mathcal{M}$.

[4]In the sequel, in order to avoid confusion, we will explicitly indicate the AMC in the satisfiability relation: e.g. $\mathcal{M}, s \models \Phi$ indicates that $s \models \Phi$ in AMC $\mathcal{M}$.

$\mathsf{T}\,\mathsf{tt}$	$\stackrel{\text{def}}{=}$	tt	$\mathsf{T}(\mathcal{S}_{\bowtie p}(\Phi))$	$\stackrel{\text{def}}{=}$	$\mathcal{S}_{\bowtie p}((\mathsf{T}\,\Phi))$
$\mathsf{T}\,\aleph$	$\stackrel{\text{def}}{=}$	$\mathcal{P}_{\geq 1}(\mathbf{X}_{\mathsf{A}(\top,\aleph)}\,\mathsf{tt})$	$\mathsf{T}(\mathcal{P}_{\bowtie p}(\varphi))$	$\stackrel{\text{def}}{=}$	$\mathcal{P}_{\bowtie p}((\mathsf{T}\,\varphi))$
$\mathsf{T}(\neg\,\Phi)$	$\stackrel{\text{def}}{=}$	$\neg(\mathsf{T}\,\Phi)$	$\mathsf{T}(\Phi\ {}_{\Delta}\mathcal{U}^{<t}\ \Phi')$	$\stackrel{\text{def}}{=}$	$(\mathsf{T}\,\Phi)\ {}_{\mathsf{A}(\Delta,\Phi)}\mathcal{U}^{<t}\ (\mathsf{T}\,\Phi')$
$\mathsf{T}(\Phi \vee \Phi')$	$\stackrel{\text{def}}{=}$	$(\mathsf{T}\,\Phi) \vee (\mathsf{T}\,\Phi')$	$\mathsf{T}(\Phi\ {}_{\Delta}\mathcal{U}^{<t}_{\Omega}\ \Phi')$	$\stackrel{\text{def}}{=}$	$(\mathsf{T}\,\Phi)\ {}_{\mathsf{A}(\Delta,\Phi)}\mathcal{U}^{<t}_{\mathsf{A}(\Omega,\Phi)}\ (\mathsf{T}\,\Phi')$

Table 13: Logic Translation function

Function A produces the action set $\mathsf{A}(\Delta,\Phi)$, over ACT_F, corresponding to formula Φ and action specifier Δ. It is defined in the expected way; $\mathsf{A}(\Delta,\Phi)$ is the set:

$$\mathsf{A}(\Delta,\Phi) \stackrel{\text{def}}{=} \{\mathsf{cod}(\gamma,p) \in ACT_F \mid \gamma \in (\Gamma\,\Delta), p \in (\mathsf{R}\,\Phi)\}$$

It is easy to see that the following lemma holds:

Lemma 4.4 *For all $\mathcal{M}$, states s of $\mathcal{M}$, $\gamma \in ACT$, and* MoSL$^-$ *formaule Φ,Δ the following holds: $\mathsf{cod}(\gamma,(\chi\,s)) \in \mathsf{A}(\Delta,\Phi)$ implies $\mathcal{M},\gamma,[] \models \Delta$.* □

An immediate consequence of Lemma 4.3, and of the definitions of functions A and cod is the following lemma:

Lemma 4.5 *For all $\mathcal{M}$, states s of $\mathcal{M}$, $\gamma \in ACT$, and* MoSL$^-$ *formulae Φ and Δ such that $\mathcal{M},\gamma,[] \models \Delta$ and $\mathcal{M},s \models \Phi$, the following holds: $\mathsf{cod}(\gamma,(\chi\,s)) \in \mathsf{A}(\Delta,\Phi)$.* □

The three lemmata above are very useful for proving the correctness of the translation function T which is defined in Table 13.

Function T essentially moves every requirement on states $\aleph$ which is an atomic proposition—or a boolean combination thereof—to a requirement on *all* transitions emanating from such states. Intuitively, it is required that the labels of such transitions are "marked" by the requirement. Technically, this is achieved by requiring them to be elements of $\mathsf{A}(\top,\aleph)$ and it is the logic counterpart of the definition of $\mapsto_F$. Notice that we use the fact that the aCSL path operator $\mathcal{P}_{\geq 1}(\cdot)$ expresses the CTL path-quantifier $\forall\cdot$. Such correspondence is justified only under specific fairness conditions [7]. On the other hand, the specific form of formulae we are dealing with (i.e. $\mathbf{X}_{\mathsf{A}(\top,\aleph)}\,\mathsf{tt}$), together with the fact that, by construction of $\mathcal{M}_F$, either *all* transitions emanating from a state have their labels included in $\mathsf{A}(\top,\aleph)$ or *none* of them, make the fairness constraints irrelevant for the case at hand. The only other interesting cases of the definition of T are those for the until formulae. Notice that action requirements Δ and Ω are enriched with those coming from formula Φ. In particular, this holds for Ω due to the fact that state properties are moved *forward* to emanating transitions. Notice finally that the complexity of the translation is linear in the size of the input formula Φ.

The following theorem guarantees the correctness of the translation:

Theorem 4.6 *For all* StoKlaim *specifications $(\beta_0,N_0,\vec{D})$,* MoSL$^-$ *formulae Φ, and states s of $\mathsf{AMC}(\beta_0,N_0,\vec{D})$ the following holds:*
$\mathsf{AMC}(\beta_0,N_0,\vec{D}),s \models_{\text{MoSL}^-} \Phi$ iff $\mathsf{FAMC}(\beta_0,N_0,\vec{D}),s \models_{aCSL} (\mathsf{T}\,\Phi)$. □

We close this section with a discussion on how the translation procedure could be extended in order to cope with action specifiers with binding capabilities. A first observation is that, broadly speaking, formulas with binding action specifiers can be thought of as proper disjunctions where

the disjuncts are obtained by different instantiations of the binders, as the following example shows. Consider the formula $\mathcal{P}_{>0.7}(\mathbf{X}^{<t}_{\{i:\mathbf{O}(!x,i)\}}\ \langle x\rangle@i)$, for some $i \in \mathcal{I}$ and $t \in \mathbb{R}_{>0}$, and suppose that the only actions of the form $(i, \mathbf{o}(v, i))$, with $v \in \mathcal{V}$, which label transitions of $\mathcal{M}$ are $(i, \mathbf{o}(v_1, i))$ and $(i, \mathbf{o}(v_2, i))$ for distinct v_1 and v_2 in $\mathcal{V}$. From the definition of the satisfiability relation of the until operator it is clear that, under the above assumption, for any state s of $\mathcal{M}$, the sets

$$\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{i:\mathbf{O}(v_1,i)\}}\ \langle v_1\rangle@i\}$$

and

$$\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{i:\mathbf{O}(v_2,i)\}}\ \langle v_2\rangle@i\}$$

are *disjoint* and their *union* coincides with the set

$$\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{i:\mathbf{O}(!x,i)\}}\ \langle x\rangle@i\}.$$

From Probability Theory we can thus conclude that

$$\mathbb{P}\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{i:\mathbf{O}(!x,i)\}}\ \langle x\rangle@i\} =$$
$$\mathbb{P}\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{i:\mathbf{O}(v_1,i)\}}\ \langle v_1\rangle@i\} + \mathbb{P}\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{i:\mathbf{O}(v_2,i)\}}\ \langle v_2\rangle@i\}$$

Notice that the fomulae in the above two sets belong to MoSL$^-$. Recall moreover that most aCSL model-checkers, like ETMCC, when applied to formulas of the form $\mathcal{P}_{\bowtie p}(\varphi)$, explicitly provide, as part of their standard result, the specific value of $\mathbb{P}\{\pi \in \mathit{Paths}(s) \mid \pi \models \varphi\}$ for each state s of the model AMC. Consequently we can apply function T to each of the above two formulae and use the resulting aCSL ones for getting the probabilities of the above two sets automatically, by means of (two separate sessions of) stochastic model-checking. The final step for checking if the original formula holds will be to compare the sum of the two resulting values against 0.7.

In the remainder of this section we will describe in more detail under which conditions and how we can proceed for generic formulae of the form $\mathcal{P}_{\bowtie p}(\mathbf{X}^{<t}_{\Omega}\ \Phi)$.

We say that action specifier ξ is *closed* if and only if every occurrence of any variable in ξ is preceded by the exclamation mark. In other words, a closed action specifier is one whose variables are only binders. The following proposition easily follows from the relevant definitions:

Proposition 4.7 *For all states s of $\mathcal{M}$, closed action specifiers ξ, state formula Φ, and substitutions Θ and Θ' such that $(\mathit{dom}\,\Theta) = (\mathit{dom}\,\Theta') = (\mathit{BS}\,\xi)$ and $[] \neq \Theta \neq \Theta' \neq []$, the two sets $\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{\xi\Theta\}}\ \Phi\Theta\}$ and $\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{\xi\Theta'\}}\ \Phi\Theta'\}$ are disjoint.* □

Given action specifier ξ, we let $(\mathsf{SUB}\,\xi)$ be the set of all substitutions Θ such that $s \overset{\gamma,\lambda}{\longmapsto} s'$ is a transition of $\mathcal{M}$ and $\mathit{match}(\xi, \gamma, \beta_0) = \Theta$. Notice that $(\mathsf{SUB}\,\xi)$ is always finite since $\mathcal{M}$ is finite. It is easy to see that the following proposition holds:

Proposition 4.8 *For all states s of $\mathcal{M}$, closed action specifiers ξ, state formula Φ, the set $\{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{\xi\}}\ \Phi\}$ is equal to*

$$\bigcup_{\substack{\Theta \in (\mathit{SUB}\,\xi),\\ \Theta \neq []}} \{\pi \in \mathit{Paths}(s) \mid \pi \models \mathbf{X}^{<t}_{\{\xi\Theta\}}\ \Phi\Theta\}$$

□

The above considerations can be generalized to the case $\mathbf{X}^{<t}_{\Omega}\ \Phi$, provided Ω satisfies certain constraints. We say that action set $\Omega = \{\xi_1, \ldots, \xi_n\}$ is *non-redundant* if and only if for all pairs of

distinct elements ξ_j and ξ_k there exist no substitutions Θ_j and Θ_k such that $(\mathsf{dom}\,\Theta_j) = (\mathsf{BS}\,\xi_j)$, $(\mathsf{dom}\,\Theta_k) = (\mathsf{BS}\,\xi_k)$, and $\xi_j\Theta_j = \xi_k\Theta_k$. The following are examples of non-redundant action sets:

$$\{!z_1 : \mathtt{o}(!x, !z_2), !z_1 : \mathtt{i}(!x, !z_2)\}$$

$$\{!z_1 : \mathtt{n}(!z_2), !z_1 : \mathtt{o}(!z_2, i)\}$$

$$\{i_1 : \mathtt{o}(!x, i_2), i_1 : \mathtt{o}(!x, i_3)\} \text{ with } i_2 \neq i_3.$$

On the other hand, set $\{i_1 : \mathtt{o}(!z_1, !z_2), i_1 : \mathtt{o}(!z_2, !z_1)\}$ is clearly *not* non-redundant, since, for instance, $(i_1 : \mathtt{o}(!z_1, !z_2))[i_1/z_1, i_1/z_2] = (i_1 : \mathtt{o}(!z_2, !z_1))[i_1/z_1, i_1/z_2]$. We say that action set $\Omega = \{\xi_1, \ldots, \xi_n\}$ is *closed* if and only if all of its elements are closed. The following proposition easily follows from the relevant definitions and the propositions above:

Proposition 4.9 *For all states s of $\mathcal{M}$, state-formulae Φ, $t \in \mathbb{R}$, and non-redundant, closed, action sets $\Omega = \{\xi_1, \ldots, \xi_n\}$, the following holds:*

i) for $j \neq k$, the two sets $\{\pi \in Paths(s) \mid \pi \models \mathbf{X}^t_{\xi_j}\, \Phi\}$ and $\{\pi \in Paths(s) \mid \pi \models \mathbf{X}^t_{\xi_k}\, \Phi\}$ are disjoint*; and*

ii) the set $\{\pi \in Paths(s) \mid \pi \models \mathbf{X}^t_{\Omega}\, \Phi\}$ is equal to

$$\bigcup_{j=1}^{n} \{\pi \in Paths(s) \mid \pi \models \mathbf{X}^{<t}_{\xi_j}\, \Phi\}$$

□

Putting everything together, we get the following

Proposition 4.10 *For all states s of $\mathcal{M}$, state-formulae Φ, $t \in \mathbb{R}$, and non-redundant, closed, action sets $\Omega = \{\xi_1, \ldots, \xi_n\}$, the set $\{\pi \in Paths(s) \mid \pi \models \mathbf{X}^t_{\Omega}\, \Phi\}$ is equal to*

$$\bigcup_{j=1}^{n} \left(\bigcup_{\substack{\Theta \in (\mathsf{SUB}\,\xi_j),\\ \Theta \neq []}} \{\pi \in Paths(s) \mid \pi \models \mathbf{X}^{<t}_{\xi_j\Theta}\, \Phi\Theta\} \right)$$

where all *sets in the union are* mutually disjoint *and $\mathbf{X}^{<t}_{\xi_j\Theta}\, \Phi\Theta \in \text{MoSL}^{-}$.* □

The direct consequence of the above proposition is that in order to check if $s \models \mathcal{P}_{\bowtie p}(\mathbf{X}^t_{\Omega}\, \Phi)$ one can (i) generate substitutions $\Theta \in (\mathsf{SUB}\,\xi_j)$ by means of automatic inspection of $\mathcal{M}$; (ii) use translation T and a model-checker for aCSL for computing $\mathbb{P}\{\pi \in Paths(s) \mid \pi \models \mathbf{X}^{<t}_{\xi_j\Theta}\, \Phi\Theta\}$ separately for each of such sets of paths, (iii) and then check if the sum p' of the results thus obtained is $\bowtie p$. This obviously requires minor modifications to the scripts of the model-checker, but without changing its main functionalities. The case $\Omega = \top$ can be dealt with similarly, since $\mathcal{M}$ is finite.

It is finally worth pointing out that the above technique cannot be easily upgraded for dealing with generic until formulae. The reason is that for generic until formulae the intersections of those sets of paths generated from different (instantiations of) action specifiers may be nonempty. Consider for instance the formula $\Phi\ {}_{\top}\mathcal{U}_{\{\xi_1,\xi_2\}}\ \Psi$ where $\xi_1 = i_1 : \mathtt{n}(!z)$ and $\xi_2 = i_2 : \mathtt{n}(!z)$ with $i_1 \neq i_2$. Take any path $\pi = s_0(\gamma_0, t_0)s_1(\gamma_1, t_1)s_2(\gamma_2, t_2)\pi'$ such that $s_0 = s$, $s_0 \models \Phi$,

$$
\begin{array}{rcl}
V & \triangleq & ((\mathbf{out}(V)@north, rn).\mathbf{nil}) + \\
 & & ((\mathbf{out}(V)@south, rs).\mathbf{nil}) + \\
 & & ((\mathbf{out}(V)@east, re).\mathbf{nil}) + \\
 & & ((\mathbf{out}(V)@west, rw).\mathbf{nil}) \\
 & & \\
O_{jk} & \triangleq & ((\mathbf{in}(!X)@\mathsf{self}, u_{jk}).(\mathbf{eval}(X)@\mathsf{self}, r_{jk}).O_{jk}) + \\
 & & ((\mathbf{in}(!X)@\mathsf{self}, d_{jk}).O_{jk})
\end{array}
$$

Figure 1: Specification of an infected network

$match(\xi_1, \gamma_0, \beta_{s_0}) = [i'_1/z]$ for some $i'_1 \in \mathcal{I}$, $s_1 \models \Phi \wedge (\Psi[i'_1/z])$, $match(\xi_2, \gamma_1, \beta_{s_1}) = [i'_2/z]$ for some $i'_2 \in \mathcal{I}$, and $s_2 \models \Psi[i'_2/z]$. It is easy to see that such a π, among others, is an element of *both* set

$$\{\pi \in Paths(s) \mid \pi \models \Phi \ {}_{\top}\mathcal{U}_{\{\xi_1\}} \ \Psi\}$$

and set

$$\{\pi \in Paths(s) \mid \pi \models \Phi \ {}_{\top}\mathcal{U}_{\{\xi_2\}} \ \Psi\}$$

In the general case, simple characterizations of such intersections by means of MoSL formulae is not easy to find and we leave it for further study. On the other hand, in the case of $\mathbf{X}^t_\Omega \Phi$ the different, disjoint, sets are characterized by the *first* action of the paths, which in turn corresponds to different (instantiations of) action specifiers of (closed and non-redundant) set Ω. In any case, it should be pointed out that the number of substitutions to be generated, and, consequently, the number of model-checking subsessions which is required, may grow quite fast. For this reason, another alternative which we plan to investigate in the future is the adaptation of stochastic model checking algorithms in order to directly cope with the action binding capabilities required by MoSL.

5 Modeling and analysis of the spreading of a virus

In this section we give examples of interesting qualitative and quantitative properties of a virus diffusion model which can be expressed in MoSL and we show the result of model-checking their translations with ETMCC. The example we present has been inspired by a similar one in [22]. The model we use is described in detail in [16].

We model a network as a set of sites and the virus running on a site can move arbitrarily from the current site to a subset of adjacent sites, infecting them. At each site, an instance of the operating system runs, which, upon receiving the virus, can either run it or suppress it. In this paper, for the sake of simplicity, we consider simple networks which are in fact grids of $n \times m$ sites. Each site is connected with its four neighbors ($north, south, east, west$), except for border sites, which lack some connections in the obvious way (e.g. the sites on the east border have no east connection). Moreover, we assume that the virus can move only to *one* adjacent site. Finally, we refrain from modeling aspects of the virus other than the way it replicates in the network. In particular we do not consider the local effects of the virus and we make the virus die as soon as it has infected one of the neighbors of its site. Similarly, we abstract from all details of the operating system, except those directly dealing with the virus.

The process definitions for the virus V and the operating system running at each node are given in Fig. 1. For the verification experiments, we chose $n = m = 3$ in order to be able to generate the resulting LTS without using automatic tools. The network component N_0 of the initial configuration is the following:

$$i_{11} ::_{\rho_{11}} \langle V \rangle \parallel (\Big\|_{\substack{1 \le j \le 3 \\ 1 \le k \le 3}} i_{jk} ::_{\rho_{jk}} O_{jk})$$

We performed different verification sessions using different initial rate-mappings β_0, as described in the sequel. The LTS is not shown for space reasons; it consists of 28 states and 52 transitions.

There are several interesting issues of the spreading of the virus which can be addressed using MoSL. Let us consider the property, Φ_1, stating that the probability that the infection develops (i.e. the virus is running) at site i_{jk}, within t time-units after site i_{11} has been infected, is smaller than a given upper bound p. This property becomes more interesting when we define the rates associated to the detection (resp. lack of detection) of the virus in such a way that the operating systems of the sites on the diagonal from bottom-left to top-right—O_{31}, O_{22}, and O_{13}—have a relatively high rate of detection and can be considered as a firewall to protect the sites i_{32}, i_{33}, and i_{23}.

The property can be expressed in MoSL for site i_{33} and $p = 0.2$ as follows:

$$\mathcal{P}_{\leq 0.2}(\neg(V@i_{33}) \ {}_{\top}\mathcal{U}^{\leq t} \ V@i_{33})$$

We can translate it into an aCSL formula for ETMCC model checking. To that purpose, let $\aleph_1$ stand for $V@i_{33}$ and assume it be represented by q. Let also $VrsAct$ be the set of actions of the AMC associated to the specification, and assume the encoding be defined simply as[5] $\mathsf{cod}(\gamma, z) \stackrel{\text{def}}{=} (\gamma, z)$. The translation $\mathsf{T}(\Phi_1)$ then yields the following aCSL formula:

$$\mathcal{P}_{\leq 0.2}(\neg\Psi \ {}_{\mathsf{A}(\top,\neg\aleph_1)}\mathcal{U}^{\leq t} \ \Psi)$$

where Ψ is the formula $\mathcal{P}_{\geq 1}(\mathbf{X}_{\mathsf{A}(\top,\aleph_1)} \ \mathsf{tt})$, with $\mathsf{A}(\top, \aleph_1)$ being the set $\{(\gamma, q) \in VrsAct_F \mid \gamma \in VrsAct\}$ and, similarly, set $\mathsf{A}(\top, \neg\aleph_1)$ is the set $\{(\gamma, \bar{q}) \in VrsAct_F \mid \gamma \in VrsAct\}$. Fig. 2 shows the probability to reach, from the initial state, a state where the virus is running at site i_{33}. The measure is presented for time values ranging from 1 to 10 with $\beta_0\, rn = \beta_0\, rs = \beta_0\, re = \beta_0\, rw = 2$, $\beta_0\, r_{jk} = 2$ for $1 \leq j, k \leq 3$, $\beta_0\, d_{31} = \beta_0\, d_{22} = \beta_0\, d_{13} = 10$, and $\beta_0\, d_{ij} = 1$ otherwise, $\beta_0\, u_{31} = \beta_0\, u_{22} = \beta_0\, u_{13} = 1$, and $\beta_0\, u_{jk} = 10$ otherwise. We performed similar analyzes for different values of the detection (resp. lack of detection) rates of the firewall. In particular for d_{31}, d_{22}, d_{13} and u_{31}, u_{22}, u_{13} ranging over $[1, \ldots, 10]$, with $d_{(4-j)j} + u_{(4-j)j}$ constant for $1 \leq j \leq 3$ (and equal to 11). For the sake of readability, in Fig. 2 we show the results only for $d_{31}, d_{22}, d_{13} \in \{1, 6, 10\}$ and $u_{31}, u_{22}, u_{13} \in \{1, 5, 10\}$. The results clearly indicate, as expected, that for high detection rates the probability for site i_{33} to run the virus within a certain time interval is lower.

Stochastic model-checking permits also the verification of qualitative properties as a degenerate case of quantitative ones. For instance, an interesting property is: "whenever the infection develops at a certain a site, the virus may move to a neighbor in the next step". For instance, in the case of site i_{33} the property of interest, Φ_2, is:

$$V@i_{33} \Rightarrow \ \langle i_{33} : \mathbf{o}(V, i_{32})\rangle \, \mathsf{tt}$$

The translated formula $\mathsf{T}(\Phi_2)$ is given below, where we let ξ stand for $i_{33} : \mathbf{o}(V, i_{32})$, and $\aleph_1$ as before:

$$(\neg\mathcal{P}_{\geq 1}(\mathbf{X}_{\mathsf{A}(\top,\aleph_1)} \ \mathsf{tt})) \vee (\mathcal{P}_{>0}(\mathsf{tt} \ {}_{\varnothing}\mathcal{U}_{\mathsf{A}(\{\xi\},\mathsf{tt})} \ \mathsf{tt}))$$

Labeling set $\mathsf{A}(\{\xi\}, \mathsf{tt})$ is the set $\{(i_{33}, \mathbf{o}(V, i_{32}), q), (i_{33}, \mathbf{o}(V, i_{32}), \bar{q})\}$[6]. The model-checker shows that Φ_2 holds in every state.

6 Conclusions and Future Work

In this paper we presented MoSL, a stochastic logic for StoKlaim, which addresses both spatial and temporal notions in order to reflect both the topological structure of systems and their evolution over time in a probabilistic setting.

[5] In practice, due to lexical restrictions on action labels imposed by the implementation of ETMCC, the encodings we used in actual experiments are slightly more involving.

[6] Since ETMCC requires that the elements of action sets be element of the label set of the AMC, the actual set for $\mathsf{A}(\{\xi\}, \mathsf{tt})$ is $\{(i_{33}, \mathbf{o}(V, i_{32}), q), (i_{33}, \mathbf{o}(V, i_{32}), \bar{q})\} \cap VrsAct_F$.

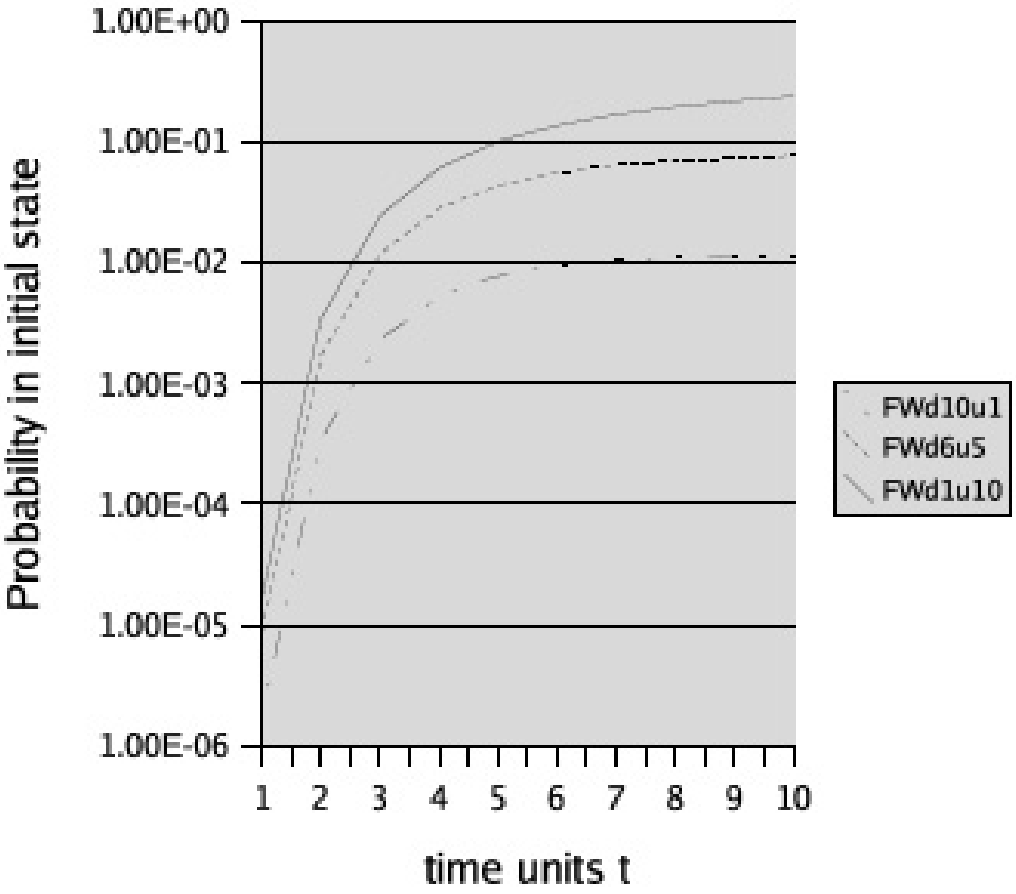

Figure 2: Results for Firewalls with different detection capability

The starting point of our proposal is to use continuous random variables with exponential distributions for modeling action durations in StoKlaim processes. In connection with such duration attributes, the logic provides probabilistic operators which naturally express steady-state probabilities as well as probability measures of paths specified with typical until formulae. The logic integrates both the state-based paradigm and the action-based one and provides specific atomic propositions addressing data and process distribution. It also provides specific atomic propositions for actions in order to characterize relevant activities taking place during executions.

The formal semantics of MoSL has been presented and a mapping from a large fragment of the logic to aCSL, the action based Continuous Stochastic Logic described in [28], has been shown and proved correct. The availability of such mapping provides the possibility of model-checking systems modelled by StoKlaim against requirements specified in MoSL using existing model-checkers for aCSL, like ETMCC.

For illustrating the technique we used a small example modeling the spreading of a virus through a network described in [16]. We analyzed some of the qualitative and quantitative aspects of this example such as the velocity of spreading of the virus.

The results in this paper show the viability of the approach and of its practical usefulness when addressing quantitative aspects of mobile systems.

The ideas proposed in this paper give rise to a whole range of related interesting research questions. First of all, proper tools for supporting system modeling and verification based on StoKlaim and MoSL should be developed.

At the logic level, we are currently investigating the inclusion in MoSL of specific operators of MoMo, the logic for Klaim presented in [20], like the *consumption* and *production* operators. The next research steps we intend to take, besides those necessary in order to keep the logic aligned with future extensions of StoKlaim, (for including features like non-exponential distributions—e.g. PH-distributions[33]—and non-determinism–e.g. moving towards Markov Decision Processes [34]) are towards implementation of the mapping to aCSL. Another approach, which we are ccurrenty investigating and which looks very promising, is the direct use of CSL model-checkers for the implementation of a model-checking algorithm for MoSL (actually for the extension of MoSL with the MoMo production and consumption operators). We also plan to investigate feasibility and convenience to develop direct model-checking algorithms.

Finally, we also plan to consider a more expressive logic that allows to reason about (spontaneous) sites failures and shall study paradigmatic examples to assess adequacy and expressiveness of the proposed logics.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science. Elsevier.*, 126:183–235, 1994.

[2] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking Continuous Time Markov Chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the Logical Characterization of Performability Properties. In U. Montanari, J. Rolim, and E. Welzl, editors, *Automata, Languages and Programming*, volume 1853 of *LNCS*, pages 780–792. Springer-Verlag, 2000.

[4] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Automated Performance and Dependability Evaluation Using Model Checking. In *Computer Performance Evaluation*, pages 261–289. Springer-Verlag, 2002.

[5] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Transactions on Software Engineering. IEEE CS*, 29(6):524–541, 2003.

[6] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In J. Baeten and S. Mauw, editors, *Concur '99*, volume 1664 of *LNCS*, pages 146–162. Springer-Verlag, 1999.

[7] C. Baier and M. Kwiatkowska. On the Verification of Qualitative Properties of Probabilistic Processes under Fairness Constraints. *Information Processing Letters*, 66(2):71–79, 1998.

[8] L. Bettini, V. Bono, R. De Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The Klaim Project: Theory and Practice. In C. Priami, editor, *Global Computing: Programming Environments, Languages, Security and Analysis of Systems*, volume 2874 of *LNCS*, pages 88–150. Springer-Verlag, 2003.

[9] L. Bettini, R. De Nicola, and M. Loreti. Formalizing Properties of Mobile Agent Systems. In F. Arbab and C. Talcott, editors, *Coordination Models and Languages*, volume 2315 of *LNCS*, pages 72–87. Springer-Verlag, 2002.

[10] L Caires and L. Cardelli. A spatial logic for concurrency (part I). *Information and Computation. Academic Press, Inc.*, 186(2):194–235, 2003.

[11] L. Cardelli. Abstractions for Mobile Computations. In J. Vitek and C. Jensen, editors, *Secure Internet Programming*, volume 1603 of *LNCS*, pages 51–94. Springer-Verlag, 1999.

[12] L. Cardelli and A. Gordon. Anytime, anywhere: modal logics for mobile ambients. In *Twentyseventh Annual ACM Symposium on Principles of Programming Languages*, pages 365–377. ACM, 2000.

[13] D. D'Aprile, S. Donatelli, and J. Sproston. CSL model checking for the GreatSPN tool. In C. Aykanat, T. Dayar, and I. Korpeoglu, editors, *Int. Symp. on Computer and Information Sciences*, volume 3280 of *LNCS*, pages 543–552. Springer-Verlag, 2004.

[14] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering. IEEE CS*, 24(5):315–329, 1998.

[15] R. De Nicola, J.-P. Katoen, D. Latella, and M. Massink. Towards a Logic for Performance and Mobility. FULL VERSION. Technical Report 2005-TR-01, Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', 2005.

[16] R. De Nicola, J.-P. Katoen, D. Latella, and M. Massink. StoKlaim: A Stochastic Extension of Klaim. Technical Report 2006-TR-01, Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', 2006. (A revised version is currently available at http://www1.isti.cnr.it/∼Latella/StoKlaim.pdf).

[17] R. De Nicola, J.-P. Katoen, D. Latella, and M. Massink. Towards a logic for performance and mobility. In A. Cerone and H. Wiklicky, editors, *Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005)*, volume 153 of *Electronic Notes in Theoretical Computer Science*, pages 161–175. Elsevier Science Publishers B.V., 2006.

[18] R. De Nicola, D. Latella, and 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, and H. Clausen, editors, *APPLIED COMPUTING 2005. Proceedings of the 20th Annual ACM Symposium on Applied Computing*, pages 428–435. Association for Computing Machinery - ACM, 2005. ISBN 1-58113-964-0.

[19] R. De Nicola and M. Loreti. A modal logic for mobile agents. *ACM Transactions on Computational Logic. ACM Press*, 5(1):79–128, 2004.

[20] R. De Nicola and 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*, volume 3657 of *LNCS*. Springer-Verlag, 2005.

[21] R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. In I. Guessarian, editor, *Proceedings of LITP Spring School on Theoretical Computer Science*, volume 469 of *LNCS*, pages 407–419. Springer-Verlag, 1990.

[22] A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic KLAIM. In R. De Nicola, G. Ferrari, and G. Meredith, editors, *Coordination Models and Languages*, volume 2949 of *LNCS*. Springer-Verlag, 2004.

[23] A. Fantechi, S. Gnesi, and G. Mazzarini. How Much Expressive Are LOTOS Expressions? In J. Quemada, J. Manas, and M. Thomas, editors, *Formal Description Techniques — III*. North-Holland Publishing Company, 1991.

[24] G. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Transactions on Software Engineering and Methodology. ACM Press*, 12(4):440–473, 2003.

[25] D. Gelernter. Generative Communication in Linda. *Communications of the ACM. ACM Press*, 7(1):80–112, 1985.

[26] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing. The International Journal of Formal Methods. Springer-Verlag*, 6(5):512–535, 1994.

[27] S. Hart and M. Sharir. Probabilistic Temporal Logics for Finite and Bounded Models. In R. De Millo, editor, *16th annual ACM symposium on Theory of computing*, pages 1–13. Association for Computing Machinery - ACM, 1984. ISBN 0-89791-133-4.

[28] H. Hermanns, J.-P Katoen, J. Meyer-Kayser, and M. Siegle. Towards Model Checking Stochastic Process Algebra. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Integrated Formal Methods - IFM 2000*, volume 1945 of *LNCS*, pages 420–439. Springer-Verlag, 2000.

[29] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Tool for Model-Checking Markov Chains. *International Journal on Software Tools for Technology Transfer. Springer-Verlag*, 4(2):153–172, 2003.

[30] V. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995.

[31] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic Symbolic Model Checking using PRISM: A Hybrid Approach. *International Journal on Software Tools for Technology Transfer. Springer-Verlag*, 6(2):128–142, 2004.

[32] S. Merz, M. Wirsing, and J. Zappe. A Spatio-Temporal Logic for the Specification and Refinement of Mobile Systems. In M. Pezzé, editor, *Fundamental Approaches to Software Engineering (FASE 2003)*, volume 2621 of *LNCS*, pages 87–101. Springer-Verlag, 2003.

[33] M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models—An Algorithmic Approach*. The Johns Hopkins University Press, Baltimore, 1981.

[34] M.L. Puterman. *Markiv Decision Processes*. The Weizmann Institute of Science, 1994.

APPENDIX: Correctness of the MoSL to aCSL translation

For the sake of notational simplicity, proofs are given only for the case of *untimed* until. The time parameter does not play any interesting role in the translation and its correctness. Moreover, the absence of absorbing nodes in the AMCs guarantees that all paths are infinite, which simplifies the definition of the satisfiability relation.

A.1 Proof of Lemma 4.3

Part (i): By induction on the structure of Φ
Case Φ of
$\mathtt{tt}$: trivial, since $\mathcal{M}, s \models \mathtt{tt}$ and $(\chi\, s) \in B(p_1, \ldots, p_n)$

$$\rightleftharpoons$$

$\aleph_j$:

$\mathcal{M}, s \models \aleph_j$

$\equiv$ {Def of χ}

$(\chi\, s) = y_1 \ldots y_n,$ with $y_j = p_j$

$\equiv$ {Def of $(\mathsf{R}\,\aleph_j)$}

$(\chi\, s) \in (\mathsf{R}\,\aleph_j)$

$$\rightleftharpoons$$

$\neg\Phi$:

$\mathsf{APC}(\neg\Phi) \ \wedge \ \mathcal{M}, s \models \neg\Phi$

$\equiv$ {Def of APC, Def of $\models$}

$\mathsf{APC}(\Phi) \ \wedge \ \neg(\mathcal{M}, s \models \Phi)$

$\equiv$ {Def of χ, I.H. }

$\mathsf{APC}(\Phi) \ \wedge \ (\chi\, s) \in B(p_1, \ldots, p_n) \ \wedge \ (\chi\, s) \notin (\mathsf{R}\,\Phi)$

$\equiv$ {Def of $\mathsf{R}(\mathtt{tt})$, Properties of Boolean Algebra}

$\mathsf{APC}(\Phi) \ \wedge \ (\chi\, s) \in (\mathsf{R}\,\mathtt{tt}) \setminus (\mathsf{R}\,\Phi)$

$\equiv$ {Def of APC, Def of R}

$\mathsf{APC}(\neg\Phi) \ \wedge \ (\chi\, s) \in (\mathsf{R}\,\neg\Phi)$

$$\rightleftharpoons$$

$\Phi \vee \Phi'$:

$\mathsf{APC}(\Phi \vee \Phi') \ \wedge \ \mathcal{M}, s \models \Phi \vee \Phi'$

$\equiv$ {Def of APC, Def of $\models$}

$\mathsf{APC}(\Phi) \ \wedge \ \mathsf{APC}(\Phi') \ \wedge \ (\mathcal{M}, s \models \Phi \ \vee \ \mathcal{M}, s \models \Phi')$

$\equiv$ {I.H. }

$\mathsf{APC}(\Phi) \ \wedge \ \mathsf{APC}(\Phi') \ \wedge$

$((\chi\, s) \in (\mathsf{R}\,\Phi) \ \vee \ (\chi\, s) \in (\mathsf{R}\,\Phi'))$

$\equiv$ {Def of APC, Set Theory, Def of R}

$\mathsf{APC}(\Phi \vee \Phi') \ \wedge \ (\chi\, s) \in \mathsf{R}(\Phi \vee \Phi')$

$$\rightleftharpoons$$

$\mathcal{S}_{\bowtie p}(\Phi), \mathcal{P}_{\bowtie p}(\Phi \ {}_{\Delta}\mathcal{U}^{<t} \ \Phi'), \mathcal{P}_{\bowtie p}(\Phi \ {}_{\Delta}\mathcal{U}^{<t}_{\Omega} \ \Phi')$: trivial, since in all these cases APC evaluates to false.

Part (ii): trivially follows from Part (i) and from the fact that if $\mathsf{APC}(\Phi) = \mathsf{ff}$ then, by definition of R,

$(\mathsf{R}\,\Phi) = (\mathsf{R}\,\mathsf{tt}) = B(p_1, \ldots, p_n)$ and, by definition of χ, $(\chi\, s) \in B(p_1, \ldots, p_n)$ Q.E.D.

A.2 Proof of Theorem 4.6

By induction on the structure of Φ, under the convention that $\mathcal{M}$ ($\mathcal{M}_F$, resp.) stands for $\mathsf{AMC}(\beta_0, N_0, \vec{D})$ ($\mathsf{FAMC}(\beta_0, N_0, \vec{D})$, resp.). For simplicity we show the proof only for the untimed cases.
Case Φ of
tt: trivial.

$\rightleftharpoons$

$\aleph_j$:

$\mathcal{M}, s \models_{\text{MoSL}^-} \aleph_j$

$\equiv$ {Lemma 4.3(i)}

$(\chi\, s) \in (\mathsf{R}\,\aleph_j)$

$\equiv$ {Def of A}

$\forall \gamma \in ACT.\ \mathsf{cod}(\gamma, (\chi\, s)) \in \mathsf{A}(\top, \aleph_j)$

$\equiv$ {Def of $\mathcal{M}_F$}

$\forall \pi_F \in Paths_F(s).\ \mathsf{ac}(\pi_F, 0) \in \mathsf{A}(\top, \aleph_j)$

$\equiv$ {Def of $\models, \mathbf{X}, \mathcal{M}_F$; fairness (Sect. 4)}

$\mathcal{M}_F, s \models_{aCSL} \mathcal{P}_{\geq 1}(\mathbf{X}_{\mathsf{A}(\top, \aleph_j)}\, \mathsf{tt})$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} (\mathsf{T}\,\aleph_j)$

$\rightleftharpoons$

$\neg\Phi$:

$\mathcal{M}, s \models_{\text{MoSL}^-} \neg\Phi$

$\equiv$ {Def of $\models$}

$\neg(\mathcal{M}, s \models_{\text{MoSL}^-} \Phi)$

$\equiv$ {I.H. }

$\neg(\mathcal{M}_F, s \models_{aCSL} (\mathsf{T}\,\Phi))$

$\equiv$ {Def of $\models$}

$\mathcal{M}_F, s \models_{aCSL} \neg(\mathsf{T}\,\Phi)$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} \mathsf{T}(\neg\Phi)$

$\rightleftharpoons$

$\Phi \vee \Phi'$:

$\mathcal{M}, s \models_{\text{MoSL}^-} \Phi \vee \Phi'$

$\equiv$ {Def of $\models$}

$(\mathcal{M}, s \models_{\text{MoSL}^-} \Phi) \vee (\mathcal{M}, s \models_{\text{MoSL}^-} \Phi')$

$\equiv$ {I.H. }

$(\mathcal{M}_F, s \models_{aCSL} (\mathsf{T}\,\Phi)) \vee (\mathcal{M}_F, s \models_{aCSL} (\mathsf{T}\,\Phi'))$

$\equiv$ {Def of $\models$}

$\mathcal{M}_F, s \models_{aCSL} ((\mathsf{T}\,\Phi) \vee (\mathsf{T}\,\Phi'))$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} \mathsf{T}(\Phi \vee \Phi')$

$\rightleftharpoons$

$\mathcal{S}_{\bowtie p}(\Phi)$:

$\mathcal{M}, s \models_{\text{MoSL}^-} \mathcal{S}_{\bowtie p}(\Phi)$

$\equiv$ {Def of $\models$}

$\lim_{t\to\infty} \mathbb{P}\{\pi \in \mathit{Paths}(s) \mid$

$\mathcal{M}, \pi(t) \models_{\text{MoSL}^-} \Phi\} \bowtie p$

$\equiv$ {I.H. }

$\lim_{t\to\infty} \mathbb{P}\{\pi_F \in \mathit{Paths}_F(s) \mid$

$\mathcal{M}_F, \pi_F(t) \models_{aCSL} (\mathsf{T}\,\Phi)\} \bowtie p$

$\equiv$ {Def of $\models$}

$\mathcal{M}_F, s \models_{aCSL} \mathcal{S}_{\bowtie p}(\mathsf{T}\,\Phi)$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} \mathsf{T}(\mathcal{S}_{\bowtie p}(\Phi))$

$\rightleftharpoons$

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

$\mathcal{M}, s \models_{\text{MoSL}^-} \mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}\ \Psi)$

$\equiv$ {Def of $\models$}

$\mathbb{P}\{\pi \in \mathit{Paths}(s) \mid$

$\exists k > 0.$
$\mathcal{M}, \mathsf{st}(\pi, k-1) \models_{\text{MoSL}^-} \Phi \wedge$
$\mathcal{M}, \mathsf{ac}(\pi, k-1), [] \models_{\text{MoSL}^-} \Omega \wedge$
$\mathcal{M}, \mathsf{st}(\pi, k) \models_{\text{MoSL}^-} \Psi \wedge$
$\forall 0 \le j \le k-2.$
$\mathcal{M}, \mathsf{st}(\pi, j) \models_{\text{MoSL}^-} \Phi \wedge$
$\mathcal{M}, \mathsf{ac}(\pi, j), [] \models_{\text{MoSL}^-} \Delta$
$\} \bowtie p$

$\Rightarrow$ {I.H., Lemma 4.5, Def of $\mathcal{M}_F$)}

$\Leftarrow$ {I.H., Lemma 4.4, Def of $\mathcal{M}_F$}

$\mathbb{P}\{\pi_F \in \mathit{Paths}_F(s) \mid$

$\exists k > 0.$
$\mathcal{M}_F, \mathsf{st}(\pi_F, k-1) \models_{aCSL} (\mathsf{T}\,\Phi) \wedge$
$\mathsf{ac}(\pi_F, k-1) \in \mathsf{A}(\Omega, \Phi) \wedge$
$\mathcal{M}_F, \mathsf{st}(\pi_F, k) \models_{aCSL} (\mathsf{T}\,\Psi) \wedge$
$\forall 0 \le j \le k-2.$
$\mathcal{M}_F, \mathsf{st}(\pi_F, j) \models_{aCSL} (\mathsf{T}\,\Phi) \wedge$
$\mathsf{ac}(\pi_F, j) \in \mathsf{A}(\Delta, \Phi)$

$\}\bowtie p$

$\equiv$ {Def of $\models$}

$\mathcal{M}_F, s \models_{aCSL} \mathcal{P}_{\bowtie p}(\mathsf{T}(\Phi)\ {}_{\mathsf{A}(\Delta,\Phi)}\mathcal{U}_{\mathsf{A}(\Omega,\Phi)}\ (\mathsf{T}\,\Psi))$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} \mathcal{P}_{\bowtie p}(\mathsf{T}(\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}\ \Psi))$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} \mathsf{T}(\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}\ \Psi))$

$\rightleftharpoons$

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

$\mathcal{M}, s \models_{\text{MoSL}^-} \mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}\ \Psi)$

$\equiv$ {Def of $\models$}

$\mathbb{P}\{\pi \in \mathit{Paths}(s) \mid \mathcal{M}, \mathsf{st}(\pi,0) \models_{\text{MoSL}^-} \Psi \vee$

$\exists k > 0.\ \mathcal{M}, \mathsf{st}(\pi,k) \models_{\text{MoSL}^-} \Psi \wedge$
$\forall 0 \le j \le k-1.$
$\mathcal{M}, \mathsf{st}(\pi,j) \models_{\text{MoSL}^-} \Phi \wedge$
$\mathcal{M}, \mathsf{ac}(\pi,j), [] \models_{\text{MoSL}^-} \Delta$
$\}\bowtie p$

$\Rightarrow$ {I.H., Lemma 4.5, Def of $\mathcal{M}_F$}

$\Leftarrow$ {I.H., Lemma 4.4, Def of $\mathcal{M}_F$}

$\mathbb{P}\{\pi_F \in \mathit{Paths}_F(s) \mid \mathcal{M}_F, \mathsf{st}(\pi_F,0) \models_{aCSL} (\mathsf{T}\,\Psi) \vee$

$\exists k > 0.\ \mathcal{M}_F, \mathsf{st}(\pi_F,k) \models_{aCSL} (\mathsf{T}\,\Psi) \wedge$
$\forall 0 \le j \le k-1.$
$\mathcal{M}_F, \mathsf{st}(\pi_F,j) \models_{aCSL} (\mathsf{T}\,\Phi) \wedge$
$\mathsf{ac}(\pi_F,j) \in \mathsf{A}(\Delta,\Phi)$
$\}\bowtie p$

$\equiv$ {Def of $\models$}

$\mathcal{M}_F, s \models_{aCSL} \mathcal{P}_{\bowtie p}((\mathsf{T}\,\Phi)\ {}_{\mathsf{A}(\Delta,\Phi)}\mathcal{U}\ (\mathsf{T}\,\Psi))$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} \mathcal{P}_{\bowtie p}(\mathsf{T}(\Phi\ {}_{\Delta}\mathcal{U}\ \Psi))$

$\equiv$ {Def of T}

$\mathcal{M}_F, s \models_{aCSL} \mathsf{T}(\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}\ \Psi))$ Q.E.D.