A Modal Logic for Mobile Agents *

Rocco De Nicola Michele Loreti

Dipartimento di Sistemi e Informatica
Università di Firenze
e-mail: {denicola, loreti}@dsi.unifi.it

Abstract

KLAIM is an experimental programming language that supports a programming paradigm where both processes and data can be moved across different computing environments. The language relies on the use of explicit localities. This paper presents a temporal logic for specifying properties of Klaim programs. The logic is inspired by Hennessy-Milner Logic (HML) and the $\mu-$calculus, but has novel features that permit dealing with state properties and impact of actions and movements over the different sites. The logic is equipped with a complete proof system that enables one to prove properties of mobile systems.

Keywords: Mobile Code Languages, Temporal Logics of Programs, Coordination Models, Proof Systems.

1 Introduction

The increasing use of wide area networks, especially the Internet, has stimulated the introduction of new programming paradigms and languages that model interactions among hosts by means of *mobile agents*; these are programs that are transported and executed on different sites. For this class of programs, like for other formalisms, it is crucial to have tools for establishing deadlock freeness, liveness and correctness with respect to given specifications. However for programs involving different actors and authorities it is also important to establish other properties such as resources allocation, access to resources and information disclosure. For this purpose, in this paper we advocate the use of temporal logics for specifying and verifying dynamic properties of mobile agents running over a wide area network.

Modal logics have been largely used as formal tools for specifying and verifying properties of concurrent systems. Properties are specified by means of *temporal* and *spatial modalities*. To verify whether a concurrent system satisfies a formula two main approaches are classically available. These are respectively based on *proof systems* and *model checking*. In the proof system approach,

*This work has been partially supported by EU within the FET – Global Computing initiative, project MIKADO IST-2001-32222 and by MIUR project NAPOLI. The funding bodies are not responsible for any use that might be made of the results presented here.

inference rules are provided in order to build proofs that establish the satisfaction of formulae by systems. In the model checking approach, instead, a methodology is introduced to verify *automatically* whether a system belongs to the formal model of a formula. In this paper, we will present a new temporal logic and develop a proof system and develop a proof system for it.

We shall first introduce a formalism for describing processes and nets, then we will define a modal logic that fits naturally with the language. Finally, we shall present the proof system. In the rest of this introduction we briefly describe the proposed framework.

KLAIM

KLAIM is based on Linda [10, 11] but makes use of *multiple* located tuple spaces and *process operators* inspired by Milner's CCS [13].

In Linda, communication is asynchronous and is performed via shared space. Messages are structured and are named *tuples*; the shared space is named *tuple space*. Linda provides two actions for inserting tuples in tuple space (**out** and **eval**) and two actions for retrieving tuples from tuple space (**in** and **read**), where elements are selected by using *pattern matching*.

KLAIM processes are built by using classic CCS operators like parallel composition, non deterministic choice and action prefixing.

In KLAIM programs, called *nets*, tuple spaces and processes are distributed over different localities $(s, s_1, \ldots)$ and the classical Linda operations are indexed with the location of the tuple space they operate on to obtain $\mathbf{out}(t)@s$, $\mathbf{in}(t)@s$, $\mathbf{read}(t)@s$, and $\mathbf{eval}(P)@s$. This allows programmers to distribute/retrieve data and processes over/from different nodes directly and thus manage the physical distribution of processes, the allocation policies, and the agents mobility.

In order to model the evolution of a KLAIM net, we define a labelled operational semantics that, differently from the original one [9], does not rely on structural congruence and makes use of explicit labels. The labels carry information about the action performed, the localities involved in the action and the transmitted information. Transition labels have the following structure:

$$\mathbf{x}(s_1, arg, s_2)$$

where **x** denotes the kind of the action performed (**o** for **out**, **i** for **in**, **r** for **read** and **e** for **eval**). Locality s_1 denotes the *place* where the action is executed, while s_2 is the locality where the action takes effect. Finally, *arg* is the argument of the action and can be either a tuple or a process. For instance, if from a process running at locality s_1 inserts a tuple t in the tuple space located at s_2, by executing $\mathbf{out}(t)@s_2$, then the net evolves with a transition whose label is $\mathbf{o}(s_1, t, s_2)$.

A Modal Logic for KLAIM

Since the language is based on process algebras, a natural candidate as a specification formalism is a temporal logic based on HML, the logic proposed by Hennessy and Milner to specify and verify properties of CCS agents [12]. However, one soon realizes that HML is not completely satisfactory.

In HML, temporal properties of processes are expressed by means of the *diamond* operator ($\langle A \rangle \phi$) indexed with a set of transition labels. A CCS process P satisfies $\langle A \rangle \phi$ if there exist a label a and a process P' such that $P \xrightarrow{a} P'$ and P' satisfies ϕ.

Since in KLAIM transition labels are more complex and involve many components, the diamond operator will be indexed with a *label predicate*. In the proposed framework, a net N satisfies a formula $\langle \mathcal{A} \rangle \phi$ if there exists a label a and a net N' such that we have: $N \succ\!\!\xrightarrow{a} N'$, a satisfies $\mathcal{A}$ and N' satisfies ϕ. A label predicate $\mathcal{A}$ is built from *basic label predicates* and *abstract actions* by using disjunction ($\cdot \cup \cdot$), conjunction ($\cdot \cap \cdot$) and difference ($\cdot - \cdot$).

Basic label predicates are used for denoting the set of all transition labels ($\circ$) and for referring to labels with specific source (the locality from which the action is performed) and/or target (the locality where the action takes place).

Abstract actions denote set of labels by singling out the kind of action performed (**out**, **in**, ...), the localities involved in the transition and the information transmitted. Abstract actions have the same structure of transition labels; but have *locality references* instead of localities and *tuple and process predicates* instead of tuples and processes. Locality references can be names (s, s_1, ...) or variables (u, u_1, u_2, ...). We also use $?u$ to indicate a sort of universal quantification over names. Process and tuple predicates are used for characterizing properties of tuples and processes involved in the transition. For instance, 1_P and 1_t denote generic processes and tuples, respectively. Predicates are introduced in order to describe patterns of tuples and possible computations. For instance,

- $\mathcal{A}_1 = \mathtt{R}(s_1, 1_t, s_2)$ is satisfied by a transition label if a process, located at s_1, retrieves generic tuple from the tuple space at s_2;
- $\mathcal{A}_2 = \mathtt{R}(?u_1, 1_t, s_2)$ is satisfied by a transition label if a process, located at a generic locality, retrieves a generic tuple from the tuple space at s_2;
- $\mathcal{A}_2 - \mathcal{A}_1$ is satisfied by a transition label if a process, that is not located at s_1, retrieves a generic tuple from the tuple space at s_2.

In our logic, there are also state formulae for specifying the distribution of resources (i.e. tuples in tuple spaces) in the system. The logic is equipped with a proof system based on tableau and it is inspired by [6, 18, 19].

The main differences of our solution with respect to the existing ones, also based on Hennessy-Milner logic, reside on the different transition labels. In the *standard* approaches, even those considering value passing [16], labels are considered *basic entities* and are characterize syntactically inside modal operators. Instead, in our approach transition labels are characterized in terms of their *properties*.

The definition of the proof system is, in some sense, standard. However, since we have explicit notion of values, proofs of *completeness* and *soundness* are more complex.

A small example: specifying access rights

To show how our logic can be used to specify formally access rights, we present a simple informal description of relative access rights of two nodes.

Let s_1 and s_2 be localities of a net, whose access policies have been set up in a such way that all processes, regardless of their site, can write on s_2 but only processes located at s_1 can read from s_2. Assume we are interested in proving that for this system *"never a process, located at a locality different from s_1, will retrieve a tuple from s_2"*

Using our logic we can formalize this property as follows:

$$\neg\mu\kappa.\langle\mathcal{A}_2 - \mathcal{A}_1\rangle\mathbf{tt} \vee \langle\circ\rangle\kappa$$

where $\mathcal{A}_1$ and $\mathcal{A}_2$ are the predicates defined above, $\circ$ is the basic label predicate denoting any transition label and μ is the recursion operator. The formula $\mu\kappa.\langle\mathcal{A}_2 - \mathcal{A}_1\rangle\mathbf{tt} \vee \langle\circ\rangle\kappa$ states that the system will perform eventually an action that satisfies the predicate $\mathcal{A}_2 - \mathcal{A}_1$. The negation ($\neg$) of the formula specifies that this will never happen and thus that it is never the case that a process which is not located at s_1 reads a tuple located at the tuple space of s_2.

Paper organization

The rest of the paper is organized as follows. Section 2, contains a brief introduction to KLAIM and its labelled operational semantics. The modal logic for KLAIM is presented in Section 3 and Section 4. Section 3 contains syntax and semantics of the proposed logics without recursion; moreover it contains the proof system and the proof of its soundness and completeness. In Section 4, we add recursion to the logic and extend the proof system to the new language establishing again soundness and completeness. Section 5 contains examples of use of our logical framework; we describe a few systems and the specification of some of their key properties. In Section 6, we highlight differences and similarities with other approaches.

2 KLAIM: syntax and semantics

KLAIM is a language designed for programming applications over wide area networks. KLAIM is based on the notion of *locality* and relies on a Linda-like communication model. Linda [5, 10, 11] is a coordination language with asynchronous communication and shared memory. The shared space is named *Tuple Space*, tuples are structured data.

In KLAIM, like in Linda, messages are structured and named *tuples*. Tuples are sequences of *actual* and *formal* fields. Actual fields are value. Formal fields are *variables* that will be assigned when a tuple is retrieved. Formal field are denoted by placing a '!' before the variable. Tuples are retrieved from tuple spaces by pattern matching.

A KLAIM system, called a *net*, is a set of *nodes*, univocally denoted by *physical names*. Every node has a computational component (a set of processes running in parallel) and a data component (a tuple space). The operations over tuple spaces take as argument the name of the node where the target tuple space resides and a tuple. $\mathcal{S}$ denotes the set of *physical name* (or *sites*) while $s, s_1, s_2, \ldots$ denote its element.

Programs refer to nodes by means of *logical localities*. Logical localities can be thought of as aliases for physical names. The set *Loc* is used to denote the set of logical localities and l, l_1, $l_2, \ldots$ to denote its elements. The distinct logical locality `self` $\in$ *Loc* is used to refer the node

where programs are running. In KLAIM every node is also equipped with an *allocation environment* (ρ) which maps logical localities to physical localities. Every program binds logical localities to physical localities by using allocation environment of the node where it is running.

Set	Elements	Description
$\mathcal{S}$	s	*Physical localities* or *Sites*
Loc	l	*Logical localities*
$VLoc$	u	*Locality variables*
$VLoc \cup Loc \cup \mathcal{S}$	ℓ	
$VLoc \cup \mathcal{S}$	σ	
Val	v	*Basic values*
Var	x	*Value variables*
Exp	e	*Expressions*
Ψ	A	*Process identifiers*
χ	X	*Process variables*
$\mathcal{R} \subseteq Loc \rightarrow \mathcal{S}$	ρ	*Allocation Environments*
Net	N	KLAIM *nets*
$Proc$	P	KLAIM *processes*
Act	act	KLAIM *actions*
$\mathcal{T}$	t	*Tuples*
	et	*Evaluated tuples*
$Subst$	δ	*Substitutions*

Table 1: KLAIM syntactic categories

The syntax of KLAIM nets is defined in Table 2. A node is defined by three parameters: the physical name s, the allocation environment ρ and the running process P that also contains information about the local tuple space, i.e. the set of tuples residing at s. A net N can be obtained via the parallel composition of nodes.

N	$::=$	$s ::_\rho P$	(node)
	$\mid$	$N_1 \parallel N_2$	(net composition)

Table 2: Net syntax

Let $VLoc$ and χ be, respectively, sets of locality and process variables, which are ranged over by u and X. Elements of $\mathcal{S} \cup Loc \cup VLoc$ are denoted by ℓ while those of $VLoc \cup \mathcal{S}$ will be denoted by σ. The set of basic values v, v_1, ... will be denoted by Val while the set of variables x, x_1, ... will be denoted by Var. Here, we do not specify what the basic values are. Finally, Ψ is the set of process identifiers A. The KLAIM syntactic categories are summarized in Table 1.

Process syntax is defined in Table 3, where **nil** stands for the process that cannot perform any actions, $P_1|P_2$ and $P_1 + P_2$ stand for the parallel and nondeterministic composition of P_1 and P_2, respectively. The term $act.P$ stands for the process that executes the action act then behaves like P. The possible actions are: $\mathbf{out}(t)@\ell$, $\mathbf{in}(t)@\ell$, $\mathbf{read}(t)@\ell$, $\mathbf{eval}(P)@\ell$ and $\mathbf{newloc}(u)$.

$$
\begin{array}{lcll}
P & := & \mathbf{nil} & \text{(null process)} \\
 & | & act.P & \text{(action prefixing)} \\
 & | & \mathbf{out}(et) & \text{(evaluated tuple process)} \\
 & | & P_1 \mid P_2 & \text{(parallel composition)} \\
 & | & P_1 + P_2 & \text{(nondet. choice)} \\
 & | & X & \text{(process variable)} \\
 & | & A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle & \text{(process invocation)} \\
act & ::= & \mathbf{out}(t)@\ell \mid \mathbf{in}(t)@\ell \mid \mathbf{read}(t)@\ell \mid \mathbf{eval}(P)@\ell & \\
 & | & \mathbf{newloc}(u) & \\
t & ::= & f \mid f, t & \\
f & ::= & v \mid P \mid \ell \mid !\,x \mid !\,X \mid !\,u &
\end{array}
$$

Table 3: Process Syntax

$$
\begin{array}{llllll}
\mathcal{T}[\![\, v \,]\!]_\rho = v & \mathcal{T}[\![\, P \,]\!]_\rho = P\{\rho\} & \mathcal{T}[\![\, \ell \,]\!]_\rho = \rho(\ell) \\
\mathcal{T}[\![\, !\,x \,]\!]_\rho = !\,x & \mathcal{T}[\![\, !\,X \,]\!]_\rho = !\,X & \mathcal{T}[\![\, !\,u \,]\!]_\rho = !\,u \\
 & \mathcal{T}[\![\, f, t \,]\!]_\rho = \mathcal{T}[\![\, f \,]\!]_\rho, \mathcal{T}[\![\, t \,]\!]_\rho &
\end{array}
$$

Table 4: Tuple Evaluation Function

The **out** action adds the result of evaluating t to the tuple space at site ℓ. A tuple is a sequence of *actual* and *formal* fields. Actual fields are value (v, ℓ or P). Formal fields are obtained by placing '!' before value, locality and process variables ($!x$, $!u$, $!X$).

The evaluation rules for tuples are given in Table 4, where $P\{\rho\}$ denotes the closure of P with respect to the allocation environment ρ, i.e. the process obtained from P by replacing every occurrence of each logical locality l with the physical locality $\rho(l)$. We will use et for denoting a *fully evaluated* tuple, i.e. tuples that, for each allocation environment ρ, $\mathcal{T}[\![\, et \,]\!]_\rho = et$.

The actions $\mathbf{in}(t)$ and $\mathbf{read}(t)$ are used to retrieve information from tuple spaces. Differently from **out** these are blocking operations; the computation is blocked until the required action can be performed, i.e. a tuple matching t can be found.

The matching predicate is defined in Table 5. Basically, two values match if they are identical while formal fields match any field of the same type. Two tuples (t_1, t_2) and (t_3, t_4) match if t_1 matches t_3 and t_2 matches t_4. The predicate match is also commutative: if t_1 matches t_2 then t_2 matches t_1. For instance $(1, !x, \mathbf{out}(3)@s.\mathbf{nil})$ matches $(1, 4, !X)$, while $(1, !x, 5)$ does not match $(!x, \mathbf{nil}, s)$.

The $\mathbf{in}(t)@\ell$ action looks for a tuple inside the tuple space at ℓ that satisfies the *matching predicate* defined in Table 5. If this tuple et exists then it is removed from the tuple space and the continuation process is closed with respect to the substitution $\delta = [et/t]$ that replaces each formal variable in t with the corresponding value in et. For instance, if $t = (s_1, !x, !X)$ and $et = (!u, 4, \mathbf{out}(3)@s_2.\mathbf{nil})$ then $[t/et] = [4/x, \mathbf{out}(3)@s_2.\mathbf{nil}/X]$.

The **read** operation behaves like **in** but it doesn't remove the tuple. The actions $\mathbf{in}(t)@\ell.P$ and $\mathbf{read}(t)@\ell.P$ act as a binder for variables in the formal fields of t. A variable is *free* if and only if it is not bound. A process P is closed if and only if each variable in P is not *free*.

The primitive $\mathbf{eval}(P)@\ell$ spawns a process P at the site ℓ. The action $\mathbf{newloc}(u)$ creates a new node and binds the variable u to its new/fresh *name* s. Prefix $\mathbf{newloc}(u).P$ binds the locality variable u in P. In the rest of the paper, we shall only consider closed processes.

Process identifiers are used in recursive process definitions. It is assumed that each process identifier A has a single defining equation $A(\widetilde{X}, \widetilde{u}, \widetilde{x}) \stackrel{def}{=} P$. All free variables in P are contained in $\widetilde{X}$, $\widetilde{u}$ and $\widetilde{x}$ and all occurrences of process identifiers in P are guarded, i.e. under the scope of an action prefixing operator. In order to avoid name clash we will assume that variable used as parameters for process identifiers are not used as *formal field* in any tuples in their definitions.

$$match(P, P) \qquad match(s, s) \qquad match(e, e)$$

$$match(!\,X, P) \qquad match(!\,u, s) \qquad match(!\,x, e)$$

$$\frac{match(et_2, et_1)}{match(et_1, et_2)} \qquad \frac{match(et_1, et_2) \quad match(et_3, et_4)}{match((et_1, et_3), (et_2, et_4))}$$

Table 5: The Matching Rules

In KLAIM tuples are modelled as processes; a tuple et is in the tuple space of a node s if and only if in s there is a process $\mathbf{out}(et)$. Moreover it is supposed that $\mathbf{out}(et)$ never appears in the scope of an action prefixing. We will use $\mathcal{T}$ for the set of tuples, *Proc* for the set of KLAIM processes

and *Act* for the set of KLAIM actions.

2.1 Operational Semantics

In this section, we present a labelled operational semantics for KLAIM. The original operational semantics [9] was unlabelled. The semantics proposed here is labelled and does not rely on structural congruence. This simplifies its use as a model for the logic. The semantics of KLAIM is given in two steps. The first one captures availability of resources and the resources requests put forward by processes. The second one describes the actual use of resources. We shall let *ALab* and *Etl* denote the sets of labels a and e defined by the following grammars:

$$a ::= \mathbf{o}(s_1, et, s_2) \mid \mathbf{i}(s_1, et, s_2) \mid \mathbf{r}(s_1, et, s_2) \mid \mathbf{e}(s_1, P, s_2) \mid \mathbf{n}(s_1, -,)s_2$$
$$e ::= a \mid et@s \mid \rho@s$$

We will refer to s_1 as $\mathtt{source}(a)$ and to s_2 as $\mathtt{target}(a)$. The first level of the operational semantics is defined by using the transition $\twoheadrightarrow \subseteq \mathit{Net} \times \mathit{Etl} \times \mathit{Net}$ which is the least relation defined in Table 6. Symmetric rules for $|$, $+$ and $\|$ have been omitted. To contrast the transition of the first level with those of the second one, we shall call *local* the former and *global* the latter.

$$\text{(LTUPLE)}\ \frac{}{s ::_\rho \mathbf{out}(et) \xrightarrow{et@s} s ::_\rho \mathbf{nil}} \qquad \text{(LSITE)}\ \frac{}{s ::_\rho P \xrightarrow{\rho@s} s ::_\rho P}$$

$$\text{(LOUT)}\ \frac{}{s ::_\rho \mathbf{out}(t)@\ell.P \xrightarrow{\mathbf{o}(s, \mathcal{T}[\![t]\!]_\rho, \rho(\ell))} s ::_\rho P} \qquad \text{(LEVAL)}\ \frac{}{s ::_\rho \mathbf{eval}(Q)@\ell.P \xrightarrow{\mathbf{e}(s, Q, \rho(\ell))} s ::_\rho P}$$

$$\text{(LIN)}\ \frac{}{s ::_\rho \mathbf{in}(t)@\ell.P \xrightarrow{\mathbf{i}(s, \mathcal{T}[\![t]\!]_\rho, \rho(\ell))} s ::_\rho P} \qquad \text{(LREAD)}\ \frac{}{s ::_\rho \mathbf{read}(t)@\ell.P \xrightarrow{\mathbf{r}(s, \mathcal{T}[\![t]\!]_\rho, \rho(\ell))} s ::_\rho P}$$

$$\text{(LNEWLOC)}\ \frac{s' = sup(\mathbf{succ}(s), s ::_\rho P')}{s ::_\rho \mathbf{newloc}(u).P \xrightarrow{\mathbf{n}(s, -, s')} s ::_\rho P[s'/u]} \qquad \text{(LCALL)}\ \frac{s ::_\rho P[\widetilde{P}/\widetilde{X}, \widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}] \xrightarrow{a} N'}{s ::_\rho A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle \xrightarrow{a} N'}\ A(\widetilde{X}, \widetilde{u}, \widetilde{x}) \stackrel{def}{=} P$$

$$\text{(LCHOICE)}\ \frac{s ::_\rho P_1 \xrightarrow{a} s ::_\rho P_1'}{s ::_\rho P_1 + P_2 \xrightarrow{a} s ::_\rho P_1'}$$

$$\text{(LPARN)}\ \frac{N_1 \xrightarrow{a} N_1' \quad a \neq \mathbf{n}(s_1, -, s_2)}{N_1 \parallel N_2 \xrightarrow{a} N_1' \parallel N_2} \qquad \text{(LPARP)}\ \frac{s_1 ::_\rho P_1 \xrightarrow{a} s_1 ::_\rho P_1' \ a \neq \mathbf{n}(s_1, -, s_2)}{s_1 ::_\rho P_1 | P_2 \xrightarrow{a} s_1 ::_\rho P_1' | P_2}$$

$$\text{(LPARNEWP)}\ \frac{s_1 ::_\rho P_1 \xrightarrow{\mathbf{n}(s_1, -, s_2)} s_1 ::_\rho P_1' \quad s_3 = sup(s_2, s_1 ::_\rho P_1 | P_2)}{s_1 ::_\rho P_1 | P_2 \xrightarrow{\mathbf{n}(s_1, -, s_3)} s_1 ::_\rho P_1'[s_3/s_2] | P_2}$$

$$\text{(LPARNEWN)}\ \frac{N_1 \xrightarrow{\mathbf{n}(s_1, -, s_2)} N_1' \quad s_3 = sup(s_2, N_1 \parallel N_2)}{N_1 \parallel N_2 \xrightarrow{\mathbf{n}(s_1, -, s_3)} N_1'[s_3/s_2] \parallel N_2}$$

Table 6: The Operational Semantics: Local Transitions

Intuitively we have that $N \xrightarrow{et@s} N'$ if there exists a tuple et in the tuple space of s while $N \xrightarrow{\rho@s} N'$ if s is a node of N with allocation environment ρ. Moreover we have a local transition for every basic action. For instance we have a transition:

$$N_1 \xrightarrow{\mathbf{e}(s_1, P, s_2)} N_2$$

if N_1 contains a process, that is located at s_1, which will call for evaluation of P at a site s_2. Analogously, we have a local transition labelled with $\mathbf{o}(s_1, t, s_2)$, $\mathbf{i}(s_1, t, s_2)$ or $\mathbf{r}(s_1, t, s_2)$ whenever there is a process, located at s_1, that wants to perform $\mathbf{out}(t)@s_2$, $\mathbf{in}(t)@s_2$ or $\mathbf{read}(t)@s_2$ respectively.

The **newloc** action is more critical; it requires handling creation of new sites. To this purpose, we shall assume existence of a total ordering over sites and of a function **succ** such that $\mathbf{succ}(s)$ yields the immediate successor of site s within the given total ordering. If N is a net and s a physical locality, we write $s \in N$ if there exists a node s (or a reference to s) in N. We write $s \notin N$ if s is unknown in N. Finally, we define $sup(s, N)$ as follows:

$$sup(s, N) = \begin{cases} s & \text{if } s \notin N \\ sup(\mathbf{succ}(s), N) & \text{otherwise} \end{cases}$$

This function yields the smallest site s' that is not in N and is bigger or equal to s.

Rules (LPARNEWP) and (LPARNEWN) ensure that the name of new node will be fresh. Let us consider $N_1 \parallel N_2$, if

$$N_1 \xrightarrow{\mathbf{n}(s_1, -, s_2)} N_1'$$

and s_2 appears in N_2, a new fresh name s_3 is selected and all the references to s_2 in N_1' are replaced with references to s_3 (the node s_2 does not exists in N'). This permits guaranteeing that $N \xrightarrow{\mathbf{n}(s_1, -, s_2)} N'$ if there exists a node s_1 that would create a new node s_2 which is not in N'.

Notice that, rules (LNEWLOC), (LPARNEWP) and (LPARNEWN) permit expressing the *standard* semantics for KLAIM **newloc** without knowing the composition of all nets. We would like to remark that the same result could be obtained by using a *restriction* operator similar to the one used in π-calculus [15]. Our choice has the advantage that it permits dealing with creation new names exactly like with other actions and avoids the problems that arise when one has to consider restrictions in the logic.

$$\text{(GOUT)}\ \frac{N_1 \xrightarrow{\mathbf{o}(s_1, et, s_2)} N_1' \quad N_1' \xrightarrow{\rho@s_2} N_2}{N_1 \succ\!\xrightarrow{\mathbf{o}(s_1, et, s_2)} N_2 \parallel s_2 ::_\rho \mathbf{out}(et)} \qquad \text{(GIN)}\ \frac{N_1 \xrightarrow{\mathbf{i}(s_1, t, s_2)} N_1' \quad N_1' \xrightarrow{et@s_2} N_2 \quad match(t, et)}{N_1 \succ\!\xrightarrow{\mathbf{i}(s_1, et, s_2)} N_2[et/t]}$$

$$\text{(GEVAL)}\ \frac{N_1 \xrightarrow{\mathbf{e}(s_1, P, s_2)} N_1' \quad N_1' \xrightarrow{\rho@s_2} N_2}{N_1 \succ\!\xrightarrow{\mathbf{e}(s_1, P, s_2)} N_2 \parallel s_2 ::_\rho P} \qquad \text{(GREAD)}\ \frac{N_1 \xrightarrow{\mathbf{r}(s_1, t, s_2)} N_1' \quad N_1' \xrightarrow{et@s_2} N_2 \quad match(t, et)}{N_1 \succ\!\xrightarrow{\mathbf{r}(s_1, et, s_2)} N_1'[et/t]}$$

$$\text{(GNEWLOC)}\ \frac{N_1 \xrightarrow{\mathbf{n}(s_1, -, s_2)} N_1' \quad N_1' \xrightarrow{\rho@s_1} N_2}{N_1 \succ\!\xrightarrow{\mathbf{n}(s_1, -, s_2)} N_2 \parallel s_2 ::_{[s_2/\mathtt{self}]\cdot\rho} \mathbf{nil}}$$

Table 7: The Operational Semantics: Global Transitions

The second level of the operational semantics is defined as the least relation $\succ\!\longrightarrow \subseteq Net \times Lab \times Net$ induced by the rules in Table 7. The *global* relation $\succ\!\longrightarrow$ reflects the intuitive semantics of basic KLAIM actions.

The rule (GOUT) states that, whenever in a net N_1 a process located at s_1 wants to insert an evaluated tuple et in the tuple space located s_2:

$$N_1 \xrightarrow{\mathbf{o}(s_1, et, s_2)} N_1'$$

and, in the net, there exists a node named s_2:

$$N_1' \xrightarrow{\rho @ s_2} N_2$$

then the new elementary process $\mathbf{out}(et)$ is placed at s_2:

$$N_1 \xrightarrow{\mathbf{o}(s_1, et, s_2)} N_2 \parallel s_2 ::_\rho \mathbf{out}(et)$$

The rule (GEVAL) is very similar to the previous one. In that case, if a process located at s_1 asks for evaluating a process P at s_2:

$$N_1 \xrightarrow{\mathbf{e}(s_1, P, s_2)} N_1'$$

the new process P is activated at s_2:

$$N_1 \xrightarrow{\mathbf{e}(s_1, P, s_2)} N_2 \parallel s_2 ::_\rho P$$

The construct **eval** relies on dynamic scoping: the logical localities of P are evaluated using the allocation environment of the destination node ℓ. Conversely, evaluation of process inserted in tuple spaces via **out** relies on static scoping: in P logical localities are evaluated using the allocation environment of the source node.

Rules (GIN) and (GREAD) have the same assumptions but they differ with each other for the conclusion. The first one states that if a process located at s_1 asks for a tuple matching t from the tuple space located at s ($N_1 \xrightarrow{\mathbf{i}(s_1, t, s_2)} N_1'$) and a such tuple et there exists in s_2 ($N_1' \xrightarrow{et @ s_2} N_2$), then it is retrieved ($N_1 \xrightarrow{\mathbf{i}(s_1, et, s_2)} N_2$). In the case of (GREAD), instead, the matching tuple is not removed from the tuple space located at s_2 ($N_1 \xrightarrow{\mathbf{r}(s_1, et, s_2)} N_1'$).

Finally, the rule (GNEWLOC), which handles the creation of new names, states that whenever a process located at s_1 wants to create a new node named s_2:

$$N_1 \xrightarrow{\mathbf{n}(s_1, -, s_2)} N_1'$$

and the allocation environment of s_1 is ρ:

$$N_1' \xrightarrow{\rho @ s_1} N_2$$

then a new node named s_2 is added to the net. The allocation environment of new node is obtained from ρ by binding the logical locality `self` to s_2 ($[s_2/\texttt{self}] \cdot \rho$):

$$N_1 \xrightarrow{\mathbf{n}(s_1, -, s_2)} N_2 \parallel s_2 ::_{[s_2/\texttt{self}] \cdot \rho} \mathbf{nil}$$

We shall now present a few definitions and results that permit guaranteeing finite branching of our operational semantics.

Definition 2.1 *Let N and N' be nets we write $N \succ\!\!\rightarrow^* N'$ if and only if:*

- $N' = N$ *or* $\exists a, N'' :\ N \succ\!\!\xrightarrow{a} N'', N'' \succ\!\!\rightarrow^* N'$.

Lemma 2.2 *Let N be a net, then $\{N' | \exists e.N \xrightarrow{e}\!\!\gg N'\}$ is finite.*

Proof: The proof goes by induction on the structure of N.

Base of Induction Let $N = s ::_\rho P$. We prove this case by induction on the structure of P and assume that P_1 and P_2 are such that the set $\{N' | \exists e.s ::_\rho P_i \xrightarrow{e}\!\!\gg N'\}$ is finite.

- $P = \mathbf{nil}$: $\{N' | \exists e.s ::_\rho \mathbf{nil} \xrightarrow{e}\!\!\gg N'\}) = \{s ::_\rho \mathbf{nil}\}$
- $P = \mathbf{out}(et)$: $\{N' | \exists e.s ::_\rho \mathbf{out}(et) \xrightarrow{e}\!\!\gg N'\} = \{s ::_\rho \mathbf{out}(et), s ::_\rho \mathbf{nil}\}$
- $P = act.P_1$: $\{N' | \exists e.s ::_\rho act.P \xrightarrow{e}\!\!\gg N'\} = \{s ::_\rho P', s ::_\rho act.P_1\}$ where, let $s' = sup(\mathbf{succ}(s), s ::_\rho P)$, $P' = P_1[s/u]$ if $act = \mathbf{newloc}(u)$ and $P' = P_1$ otherwise.
- $P = P_1 | P_2$: $\{N' | \exists e.s ::_\rho P_1 | P_2 \xrightarrow{e}\!\!\gg N'\} =$

  $$\{s ::_\rho P_1' | P_2 | \exists e.s ::_\rho P_1 \xrightarrow{e}\!\!\gg s ::_\rho P_1'\} \cup \{s ::_\rho P_1 | P_2' | \exists e.s ::_\rho P_2 \xrightarrow{e}\!\!\gg s ::_\rho P_2'\} \cup \{s ::_\rho P_1 | P_2\}$$

  which is finite for inductive hypothesis.
- $P = P_1 + P_2$: $\{N' | \exists e.s ::_\rho P_1 + P_2 \xrightarrow{e}\!\!\gg N'\} =$

  $$\{s ::_\rho P_1' | \exists e.s ::_\rho P_1 \xrightarrow{e}\!\!\gg s ::_\rho P_1'\} \cup \{s ::_\rho P_2' | \exists e.s ::_\rho P_2 \xrightarrow{e}\!\!\gg s ::_\rho P_2'\}$$

  which is finite for inductive hypothesis.
- $P = A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle$. Let $A(\widetilde{X}, \widetilde{u}, \widetilde{x}) \stackrel{def}{=} P'$. Since every occurrence of A is guarded in P', this case reduces to one of those considered above.

Inductive Hypothesis Let N_1 and N_2 be such that following set is finite:

$$\{N' | \exists a.N_i \xrightarrow{a}\!\!\gg N'\}$$

Inductive Step Let $N = N_1 \parallel N_2$. We have that the set of possible next nets is:

$$\{N_1' \parallel N_2 | \exists a.N_1 \xrightarrow{e}\!\!\gg N_1'\} \cup \{N_1 \parallel N_2' | \exists a.N_2 \xrightarrow{e}\!\!\gg N_2'\}$$

which is finite for inductive hypothesis. □

Theorem 2.3 *Let N be a net, then $\{N' | \exists a.N \succ\!\!\xrightarrow{a} N'\}$ is finite.*

Proof: The claim follows directly from the above lemma. □

2.2 A small example: itinerant agent

In this section, we present the evolution of a simple net. Let N be the net defined as follows:

$$s_1 :: Proc_1 | \mathbf{out}(s_2) \parallel s_2 :: \mathbf{nil}$$

where

$$Proc_1 \stackrel{def}{=} \mathbf{in}(!u)@\mathtt{self}.\mathbf{eval}(Proc_1)@u.\mathbf{out}(\mathtt{self})@u.\mathbf{nil}$$

This process, located at s_1, retrieves a locality at `self` ($\mathbf{in}(!u)@\mathtt{self}$). Later, the process evaluates itself and inserts the locality `self` in the tuple space of the read locality ($\mathbf{eval}(Proc_1)@u.\mathbf{out}(\mathtt{self})@u.\mathbf{nil}$)

The proposed net evolves according to the operational semantics of Tables 6-7 as follows:

1. First the tuple (s_2) is retrieved from the tuple space of s_1. With a transition labelled $\mathbf{i}(s_1, (s_2), s_1)$ the net evolves in

$$N_1 = s_1 :: \mathbf{eval}(Proc_1)@s_2.\mathbf{out}(\mathtt{self})@s_2.\mathbf{nil} \parallel s_2 :: \mathbf{nil}$$

2. The process $Proc_1$ is evaluated at s_2 and, with a transition labelled $\mathbf{e}(s_1, Proc_1, s_2)$ the net evolves in

$$N_2 = s_1 :: \mathbf{out}(\mathtt{self})@s_2.\mathbf{nil} \parallel s_2 :: Proc_1$$

3. Tuples (s_1) is inserted in the tuple space of s_2 transition labelled $\mathbf{o}(s_1, s_1, s_2)$ the net evolves in

$$N_3 = s_1 :: \mathbf{nil} \parallel s_2 :: Proc_1 | \mathbf{out}(s_1)$$

4. Tuple (s_1) is retrieved from s_2, and with a transition labelled $\mathbf{i}(s_2, (s_1), s_2)$ the net evolves in

$$N_4 = s_1 :: \mathbf{nil} \parallel s_2 :: \mathbf{eval}(Proc_1)@s_1.\mathbf{out}(\mathtt{self})@s_1.\mathbf{nil}$$

5. The process $Proc_1$ is evaluated from s_2 at s_1. By a transition labelled $\mathbf{e}(s_2, Proc_1, s_1)$ the net evolves in:

$$N_5 = s_1 :: Proc_1 \parallel s_2 :: \mathbf{out}(\mathtt{self})@s_1.\mathbf{nil}$$

6. Finally, with a transition labelled $\mathbf{o}(s_2, (s_2), s_1)$ net N_5 evolves in N.

3 A Modal Logic for KLAIM: the finite fragment

In this section, we introduce a logic that allows us to specify and prove properties of mobile system specified in KLAIM. Our approach draws inspiration from Hennessy-Milner Logic [12]. In their logical framework, dynamic properties of CCS processes are captured by making use of modal operators, $\langle\rangle$ and $[]$, indexed over a set of labels denoting basic actions of processes.

As we have seen, in KLAIM labels are not basic entities but carry complex information like processes, tuples, sites, etc.. In our logic, the diamond operator $\langle\cdot\rangle$ is indexed with predicates that will be interpreted over labels, and used to specify subsets of the set of possible labels. Using this

approach, we shall be able to finitely refer to infinite set of labels. To characterize state properties of KLAIM programs, namely the presence of a tuple in a specific tuple space, we shall also introduce specific state formulae.

In this section we shall concentrate on the finite fragment of our logic, the next section will be dedicated to formalizing our results for the logic with recursion.

Syntax

We shall let $\mathcal{L}$ be the set of formulas ϕ induced by the following grammar:

$$\phi \ ::= \ \texttt{tp@}\sigma \ \Big| \ \mathbf{tt} \ \Big| \ \langle \mathcal{A} \rangle \phi \ \Big| \ \phi \vee \phi \ \Big| \ \neg \phi$$

State formulae $\texttt{tp@}\sigma$, where σ denotes either a physical locality or a locality variable (see Table 1), are used to specify the distribution of resources over the net (tuples over the different sites). A net N satisfies $\texttt{tp@}s$ if there is a tuple et in the tuple space located at s that *satisfies* $\texttt{tp}$

As usual $\mathbf{tt}$ is the formula satisfied by every net, $\phi_1 \vee \phi_2$ is the formula satisfied by all nets that satisfy either ϕ_1 or ϕ_2 while $\neg\phi$ is satisfied by every net that does not satisfy ϕ. Finally, to satisfy $\langle \mathcal{A} \rangle \phi$ a net N needs to evolve, by an action satisfying the predicate $\mathcal{A}$, in a net that satisfies ϕ.

Semantics

We shall introduce the interpretation function $\mathbb{M}[\![\cdot]\!] : \mathcal{L} \to 2^{Net}$ that, for each $\phi \in \mathcal{L}$, yields the set of nets that satisfy ϕ or, equivalently, the set of nets that are *models* for ϕ.

The function $\mathbb{M}[\![\cdot]\!]$ is defined by relying on functions $\mathbb{A}[\![\cdot]\!]$ and $\mathbb{T}[\![\cdot]\!]$ that provide the interpretations of label and tuple predicates. For each label predicate function $\mathbb{A}[\![\cdot]\!]$ yields a set of pairs *(transition label,substitution)* . $\mathbb{T}[\![\texttt{tp}]\!]$, instead, denotes the set of tuples that satisfy $\texttt{tp}$. Both $\mathbb{A}[\![\cdot]\!]$ and $\mathbb{T}[\![\cdot]\!]$ will be formally defined later.

Let ϕ be a closed formula, $\mathbb{M}[\![\cdot]\!]$ is defined inductively on the structure of ϕ as follows:

- $\mathbb{M}[\![\mathbf{tt}]\!] = Net$;
- $\mathbb{M}[\![\texttt{tp@}\sigma]\!] = \{N | \exists et. N \xrightarrow{et@\sigma} \gg N', et \in \mathbb{T}[\![\texttt{tp}]\!]\}$;
- $\mathbb{M}[\![\langle \mathcal{A} \rangle \phi]\!] = \{N | \exists a, \delta, N' : N \succ\!\!\xrightarrow{a} N',\ (a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!],\ N' \in \mathbb{M}[\![\phi\{\delta\}]\!]\}$;
- $\mathbb{M}[\![\phi_1 \vee \phi_2]\!] = \mathbb{M}[\![\phi_1]\!] \cup \mathbb{M}[\![\phi_2]\!]$;
- $\mathbb{M}[\![\neg\phi]\!] = Net - \mathbb{M}[\![\phi]\!]$;

The interpretation function respects the intuitive semantics of formulae. Every net is a *model* for $\mathbf{tt}$ while every net with a node s, whose tuple space contains a tuple t satisfying $\texttt{tp}$, is a model for $\texttt{tp@}s$. N is a model for $\langle \mathcal{A} \rangle \phi$ if there exist N' such that $N \succ\!\!\xrightarrow{a} N'$, $(a, \delta) \in \mathbb{A}[\![\mathcal{A}]\!]$ and $N' \in \mathbb{M}[\![\phi\{\delta\}]\!]$. Finally, N is a model for $\phi_1 \vee \phi_2$ if N is a model for ϕ_1 or ϕ_2 while N is a model for $\neg\phi$ if it is not a model for ϕ.

We write $N \models \phi$ if and only if $N \in \mathbb{M}[\![\phi]\!]$. Other formulae like $[\mathcal{A}]\phi$ or $\phi_1 \wedge \phi_2$ can be expressed in $\mathcal{L}$. Indeed $[\mathcal{A}]\phi = \neg\langle \mathcal{A} \rangle \neg\phi$ and $\phi_1 \wedge \phi_2 = \neg(\phi_1 \vee \phi_2)$. We shall use these derivable formulae as *macros* in $\mathcal{L}$.

In $\langle\mathcal{A}\rangle\phi$ the operator $\langle\mathcal{A}\rangle$ acts as a binder for the (locality) variables in ϕ that appear quantified in $\mathcal{A}$. As usual u is free in ϕ if there exists an instance of u in ϕ which is not bound and ϕ is closed if it does not contain free variables. Finally, the closure of ϕ with respect to substitution δ (written $\phi\{\delta\}$) is the formula ϕ' obtained from ϕ by replacing every free variable u with $\delta(u)$.

3.1 Label predicates

In this section, we present formal syntax and semantics of label predicates. These permit denoting infinite set of labels by specifying their properties. The set PLab of label predicates ($\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$, ...) is defined in Table 8.

$\mathcal{A}$	::=		(label predicates)	α ::=	(abstract actions)
		$\circ$	(all labels)	$\mathtt{O}(\mathtt{sr}_1, \mathtt{tp}, \mathtt{sr}_2)$	
	$\mid$	$\mathtt{Src}(\widehat{\mathtt{sr}})$	(source)	$\mid\ \mathtt{I}(\mathtt{sr}_1, \mathtt{tp}, \mathtt{sr}_2)$	
	$\mid$	$\mathtt{Trg}(\widehat{\mathtt{sr}})$	(target)	$\mid\ \mathtt{R}(\mathtt{sr}_1, \mathtt{tp}, \mathtt{sr}_2)$	
	$\mid$	α	(abstract actions)	$\mid\ \mathtt{E}(\mathtt{sr}_1, \mathtt{pp}, \mathtt{sr}_2)$	
	$\mid$	$\mathcal{A}_1 \cap \mathcal{A}_2$	(conjunction)	$\mid\ \mathtt{N}(\mathtt{sr}_1, -, \mathtt{sr}_2)$	
	$\mid$	$\mathcal{A}_1 \cup \mathcal{A}_2$	(disjunction)	sr ::=	(site references)
	$\mid$	$\mathcal{A}_1 - \mathcal{A}_2$	(difference)	s	(physical locality)
				$\mid\ ?u$	(quantified variable)
				$\mid\ u$	(free variable)

Table 8: Label predicate syntax

A label predicate $\mathcal{A}$ is built from *basic label predicates* and *abstract actions* by using disjunction ($\cdot \cup \cdot$), conjunction ($\cdot \cap \cdot$) and difference ($\cdot - \cdot$).

Basic label predicates are used for denoting the set of all transition labels ($\circ$) and for selecting labels basing the choice on their source ($\mathtt{Src}(\widehat{\mathtt{sr}})$) - the locality from which the action is performed - or on their target ($\mathtt{Src}(\widehat{\mathtt{sr}})$) - the locality where the action takes place.

Abstract actions (α) denote set of labels by singling out the kind (O for **out**, I for **in**, etc), the localities involved in the transition and the information transmitted. Abstract actions have the same structure of transition labels; however, there are *locality references* (sr) instead of localities and *tuple and process predicates* instead of tuples and processes. Locality references can be names (s, s_1, ...), free variables (u, u_1, ...) or quantified variables($?u$, $?u_1$, ...). Process and tuples predicates are used for characterizing properties of tuples and processes involved in the transition.

Semantics

Formal interpretation of labels predicates is defined by means of two interpretations functions: $\mathbb{R}[\![\cdot]\!]$ and $\mathbb{A}[\![\cdot]\!]$. The first takes as argument a locality reference sr (s, u or $?u$) and yields a set of pairs <*physical locality-substitution*>. The second takes a label predicate $\mathcal{A}$ and yields a set of

$$\mathbb{A}[\![\circ]\!] = Lab$$

$$\mathbb{A}[\![\mathtt{O}(\mathtt{sr}_1,\mathtt{tp},\mathtt{sr}_2)]\!] = \{(\mathbf{o}(s_1,t,s_2);\delta_1\cdot\delta_2)|(s_1;\delta_1)\in\mathbb{R}[\![\mathtt{sr}_1]\!]\ ,\ (s_2;\delta_2)\in\mathbb{R}[\![\mathtt{sr}_2]\!]\ ,\ t\in\mathbb{T}[\![\mathtt{tp}]\!]\}$$

$$\mathbb{A}[\![\mathtt{I}(\mathtt{sr}_1,\mathtt{tp},\mathtt{sr}_2)]\!] = \{(\mathbf{i}(s_1,t,s_2);\delta_1\cdot\delta_2)|(s_1;\delta_1)\in\mathbb{R}[\![\mathtt{sr}_1]\!]\ ,\ (s_2;\delta_2)\in\mathbb{R}[\![\mathtt{sr}_2]\!]\ ,\ t\in\mathbb{T}[\![\mathtt{tp}]\!]\}$$

$$\mathbb{A}[\![\mathtt{R}(\mathtt{sr}_1,\mathtt{tp},\mathtt{sr}_2)]\!] = \{(\mathbf{r}(s_1,t,s_2);\delta_1\cdot\delta_2)|(s_1;\delta_1)\in\mathbb{R}[\![\mathtt{sr}_1]\!]\ ,\ (s_2;\delta_2)\in\mathbb{R}[\![\mathtt{sr}_2]\!]\ ,\ t\in\mathbb{T}[\![\mathtt{tp}]\!]\}$$

$$\mathbb{A}[\![\mathtt{E}(\mathtt{sr}_1,\mathtt{pp},\mathtt{sr}_2)]\!] = \{(\mathbf{e}(s_1,P,s_2);\delta_1\cdot\delta_2)|(s_1;\delta_1)\in\mathbb{R}[\![\mathtt{sr}_1]\!],(s_2;\delta_2)\in\mathbb{R}[\![\mathtt{sr}_2]\!],P\in\mathbb{P}[\![\mathtt{pp}]\!]\}$$

$$\mathbb{A}[\![\mathtt{N}(\mathtt{sr}_1,-,\mathtt{sr}_2)]\!] = \{(\mathbf{n}(s_1,-,s_2);\delta_1\cdot\delta_2)|(s_1;\delta_1)\in\mathbb{R}[\![\mathtt{sr}_1]\!],(s_2;\delta_2)\in\mathbb{R}[\![\mathtt{sr}_2]\!]\}$$

$$\mathbb{A}[\![\mathtt{Src}(\widetilde{\mathtt{sr}})]\!] = \{(a;\emptyset)|\exists\delta.(\mathtt{source}(a),\delta)\in\bigcup_{\mathtt{sr}'\in\{\widetilde{\mathtt{sr}}\}}\mathbb{R}[\![\mathtt{sr}]\!].\}$$

$$\mathbb{A}[\![\mathtt{Trg}(\widetilde{\mathtt{sr}})]\!] = \{(a;\emptyset)|\exists\delta.(\mathtt{target}(a),\delta)\in\bigcup_{\mathtt{sr}'\in\{\widetilde{\mathtt{sr}}\}}\mathbb{R}[\![\mathtt{sr}]\!].\}$$

$$\mathbb{A}[\![\mathcal{A}_1\cup\mathcal{A}_1]\!] = \mathbb{A}[\![\mathcal{A}_1]\!]\cup\mathbb{A}[\![\mathcal{A}_2]\!]$$

$$\mathbb{A}[\![\mathcal{A}_1\cap\mathcal{A}_1]\!] = \{(a;\delta_1\cdot\delta_2)|(a;\delta_1)\in\mathbb{A}[\![\mathcal{A}_1]\!],(a;\delta_2)\mathbb{A}[\![\mathcal{A}_2]\!]\}$$

$$\mathbb{A}[\![\mathcal{A}_1-\mathcal{A}_2]\!] = \{(a;\delta)|(a;\delta)\in\mathbb{A}[\![\mathcal{A}_1]\!],\forall\delta'\ (a;\delta')\notin\mathbb{A}[\![\mathcal{A}_2]\!]\}$$

Table 9: Labels Predicates Interpretation

pairs $<$*transition label-substitution*$>$. Intuitively, $(a,\delta)\in\mathbb{A}[\![\mathcal{A}]\!]$ if transition label *a satisfies* $\mathcal{A}$ with respect to the substitution δ.

Definition 3.1 *We define the interpretation of* $\mathtt{sr}$*, written* $\mathbb{R}[\![\mathtt{sr}]\!]$*, as follows:*

$$\mathbb{R}[\![?u]\!] = \{(s;[s/u])|s\in Sites\}\qquad \mathbb{R}[\![u]\!] = \{(u;\emptyset)\}\qquad \mathbb{R}[\![s]\!] = \{(s;\emptyset)\}$$

Notice that, $?u$ is used like a *wild card* that can be replaced be every physical locality. This reference plays the role of *existential quantification* if it is used inside a $\langle\cdot\rangle$, while it works like an *universal quantification* when used instead $[\cdot]$.

Moreover, in $\langle\mathcal{A}\rangle\phi$, $\mathcal{A}$ acts as binder for variables in $\mathcal{A}$. Let $\mathcal{A}$ a label predicate, the set of variables $\mathtt{bv}(\mathcal{A})$ that are bound in $\mathcal{A}$ is defined in Table 10.

$$\mathtt{bv}(\circ) = \emptyset$$
$$\mathtt{bv}(\mathtt{Src}(\widetilde{\mathtt{sr}})) = \mathtt{bv}(\mathtt{Trg}(\widetilde{\mathtt{sr}})) = \bigcup_{\mathtt{sr}\in\widetilde{\mathtt{sr}}}\mathtt{bv}(\mathtt{sr})$$
$$\mathtt{bv}(\mathtt{O}(\mathtt{sr}_1,\mathtt{tp},\mathtt{sr}_2)) = \mathtt{bv}(\mathtt{I}(\mathtt{sr}_1,\mathtt{tp},\mathtt{sr}_2)) = \mathtt{bv}(\mathtt{R}(\mathtt{sr}_1,\mathtt{tp},\mathtt{sr}_2)) = \mathtt{bv}(\mathtt{sr}_1)\cup\mathtt{bv}(\mathtt{sr}_2)$$
$$\mathtt{bv}(\mathtt{E}(\mathtt{sr}_1,\mathtt{pp},\mathtt{sr}_2)) = \mathtt{bv}(\mathtt{sr}_1)\cup\mathtt{bv}(\mathtt{sr}_2)$$
$$\mathtt{bv}(\mathbf{n}(\mathtt{sr}_1,-,\mathtt{sr}_2)) = \mathtt{bv}(\mathtt{sr}_1)\cup\mathtt{bv}(\mathtt{sr}_2)$$
$$\mathtt{bv}(\mathcal{A}_1\cup\mathcal{A}_2) = \mathtt{bv}(\mathcal{A}_1)\cap\mathtt{bv}(\mathcal{A}2)$$
$$\mathtt{bv}(\mathcal{A}_1\cap\mathcal{A}_2) = \mathtt{bv}(\mathcal{A}_1)\cup\mathtt{bv}(\mathcal{A}_2)$$
$$\mathtt{bv}(\mathcal{A}_1-\mathcal{A}_2) = \mathtt{bv}(\mathcal{A}_1)$$
$$\mathtt{bv}(?u) = \{u\}$$
$$\mathtt{bv}(s) = \mathtt{bv}(u) = \emptyset$$

Table 10: Label predicate bound variables

Definition 3.2 *The interpretation of label predicates is defined in Table 9. Function $\mathbb{A}[\![\cdot]\!]$ relies on $\mathbb{P}[\![\cdot]\!]$ and $\mathbb{T}[\![\cdot]\!]$ that yield, respectively, the set of processes and of tuples satisfying process and tuple predicates.*

As an example, consider the label predicates presented in the Introduction. We have:

- $\mathbb{A}[\![\mathtt{R}(s_1, \mathtt{1_t}, s_2)]\!] = \{(\mathbf{r}(s_1, t, s_2), \emptyset) | t \in \mathcal{T}\}$
- $\mathbb{A}[\![\mathtt{R}(?u_1, \mathtt{1_t}, s_2)]\!] = \{(\mathbf{r}(s, t, s_2), [s/u_1]) | s \in \mathcal{S}, t \in \mathcal{T}\}$
- $\mathbb{A}[\![\mathtt{R}(?u_1, \mathtt{1_t}, s_2) - \mathtt{I}(s_1, \mathtt{1_t}, s_2)]\!] = \{(\mathbf{r}(s, t, s_2), [s/u_1]) | s \in \mathcal{S}, s \neq s_1, t \in \mathcal{T}\}$

Within our logic, label predicates can be used to express spatial properties, e.g. *where* actions take place.

3.2 Process and Tuple predicates

When defining label predicates, we made the use of both tuple and process predicates. These are used to characterize properties of tuples and processes involved in the actions.

Process predicates shall be used to specify the kind of accesses to the resources of the net (tuples and sites). These accesses are composed in for specifying their causal dependencies. The causal properties we intend to express for processes are those expressing properties like *"first read something and then use the acquired information in some way"*. Tuple predicates will characterize tuple pattern by specifying the relevant properties of their fields.

We shall use `PTuple` to denote the set of *tuple predicates* `tp` defined as follows:

`tp`	::=		**(tuple predicate)**
		$\mathtt{1_t}$	(generic tuple)
	|	`vp`	(value predicate)
	|	`lp`	(locality predicate)
	|	`pp`	(process predicate)
	|	$!u$	(locality formal)
	|	$!x$	(value formal)
	|	$!X$	(process formal)
	|	`tp, tp`	(composition)

The tuple predicate $\mathtt{1_t}$ is used for denoting a generic tuple (any tuple will satisfy $\mathtt{1_t}$). Value (`vp`), locality (`lp`) and process predicates (`pp`) are used for characterizing the properties of the values that compose the tuples, while formals ($!u$, $!x$ and $!X$) are used to denote formal fields of tuples. Finally, tuple predicates are composed by using comma ($\mathtt{tp}_1, \mathtt{tp}_2$). We will have that t_1, t_2 satisfies $\mathtt{tp}_1, \mathtt{tp}_2$ if t_1 satisfies $\mathtt{tp}_1$ and t_2 satisfies $\mathtt{tp}_2$.

A *locality predicate* `lp` is defined as follows:

$$
\begin{array}{lcll}
\texttt{lp} & ::= & & \textbf{(locality predicate)} \\
 & & \texttt{1}_\texttt{s} & \text{(generic locality)} \\
 & \mid & s & \text{(physical locality)} \\
 & \mid & u & \text{(locality variable)} \\
 & \mid & l & \text{(logical locality)}
\end{array}
$$

where $\texttt{1}_\texttt{s}$ is used for a generic locality, s and l refer to specific physical or logical locality, respectively, and u is a locality variable.

The *value predicates* vp are defined similarly

$$
\begin{array}{lcll}
\texttt{vp} & ::= & & \textbf{(value predicate)} \\
 & \mid & \texttt{1}_\texttt{v} & \text{(generic value)} \\
 & \mid & v & \text{(basic value)} \\
 & \mid & x & \text{(value variable)}
\end{array}
$$

we have terms for denoting: any value ($\texttt{1}_\texttt{v}$), a specific value v or a free variable u.

We shall use PLoc and PVal, respectively, for denoting the set of locality predicates and the set of value predicates.

Finally, we define the sets of process and access predicates PProc and PAcc like the set of terms pp and ap defined by the following grammar:

$$
\begin{array}{lcll}
\texttt{pp} & ::= & & \textbf{(process predicate)} \\
 & & \texttt{1}_\texttt{P} & \text{(generic process)} \\
 & \mid & \texttt{ap} \rightarrow \texttt{pp} & \text{(access requirements)} \\
 & \mid & \texttt{pp} \wedge \texttt{pp} & \text{(conjunction)} \\
 & \mid & X & \text{(process variable)} \\
\texttt{ap} & ::= & & \textbf{(access predicate)} \\
 & & \texttt{i(tp)@lp} & \textbf{(in)} \\
 & \mid & \texttt{r(tp)@lp} & \textbf{(read)} \\
 & \mid & \texttt{o(tp)@lp} & \textbf{(out)} \\
 & \mid & \texttt{e(pp)@lp} & \textbf{(eval)} \\
 & \mid & \texttt{n}(u) & \textbf{(newloc)}
\end{array}
$$

We use 1_P for a generic process, $pp_1 \wedge pp_2$ for the set of processes that *satisfy* pp_1 and pp_2. A process satisfies $ap \rightarrow pp$ if it might perform an access (i.e. an action) that satisfies ap and use the acquired information as specified by pp. We introduce an action predicate ap for every basic KLAIM action.

We will write $ap_1 =_\alpha ap_2$ if ap is equal to ap' after changing the name of the variables in their formal fields ($\alpha-$renaming). If $ap_1 =_\alpha ap_2$, we will use $[ap_2/ap_1]$, for the substitution of every *formal* variable of ap_1 with the corresponding one of ap_2: for instance $r(!u_1, P, !X_1)@1_s =_\alpha r(!u_2, P, !X_2)@1_s$ and $[r(!u_2, P, !X_2)@1_s / r(!u_1, P, !X_1)@1_s] = [u_2/u_1, X_2/X_1]$.

In process predicates of the form $ap \rightarrow pp$ the terms $i(tp)@lp$, $r(tp)@lp$ and $n(u)$ act as a binder for the *formal* variables in tp and for the variable u. A variable is free in pp (respectively tp) if it does not appear under the scope of a binder $i(tp)@lp$, $r(tp)@lp$ and $n(u)$. We will use $fv(pp)$ to denote the set of free variables in pp. Function $fv(\cdot)$ is formally defined in Table 11, where $form(tp)$ yields the set of variables that are used like formals in the tuple predicate tp.

$$
\begin{array}{l}
form(1_t) = form(vp) = form(pp) = form(lp) = \emptyset \\
form(!x) = \{x\} \\
form(!X) = \{X\} \\
form(!u) = \{u\} \\
form(tp_1, tp_2) = form(tp_1) \cup form(tp_2) \\
\hline
fv(1_t) = fv(!u) = fv(!x) = fv(!X) = \emptyset \\
fv(tp_1, tp_2) = fv(tp_1) \cup fv(tp_2) \\
fv(1_P) = \emptyset \\
fv(pp_1 \wedge pp_2) = fv(pp_1) \cup fv(pp_2) \\
fv(ap \rightarrow pp) = (fv(pp) - bv(ap)) \cup fv(ap) \\
fv(X) = \{X\} \\
fv(o(tp)@lp) = fv(i(tp)@lp) = fv(r(tp)@lp) = fv(tp) \cup fv(lp) \\
fv(e(pp)@lp) = fv(pp) \cup fv(lp) \\
fv(n(u)@lp) = \emptyset \\
fv(x) = \{x\} \\
fv(u) = \{u\} \\
fv(1_v) = fv(v) = fv(1_s) = fv(s) = fv(l) = \emptyset \\
\hline
bv(i(tp)@lp) = bv(r(tp)@lp) = form(tp) \\
bv(o(tp)@lp) = bv(e(pp)@lp) = \emptyset \\
bv(n(u)) = \{u\}
\end{array}
$$

Table 11: Free variables of process predicates

Process predicates can be thought as types that reflect the possible accesses a process might perform along its computation; they also carry information about the possible use of the acquired resources [8]. This allows us to specify sophisticated properties on process capabilities. For instance we can specify the set of processes that, after reading the name of a locality from u_1, spawn a process to the read locality:

$$i(!u)@s_1 \rightarrow e(1_P)@u \rightarrow 1_P$$

This predicate is satisfied by both $\mathbf{in}(!u_1)@s_1.\mathbf{eval}(P)@u_1.Q$ $\mathbf{in}(!u_1)@s_1.\mathbf{in}(!x)@u_1.\mathbf{eval}(P)@u_1.Q$ but it is not satisfied by $\mathbf{in}(!u_2)@s_1.\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.\mathbf{nil}$; indeed, no process is evaluated at the locality retrieved from s_1.

3.2.1 Semantics

In this section, we show how the predicates introduced in the previous section are interpreted.

Interpretation of locality, value and tuples predicates are, in same sense, obvious. The formal definition of functions $\mathbb{L}[\![\cdot]\!]$: $\texttt{PLoc} \rightarrow \mathcal{S} \cup Loc \cup VLoc$, $\mathbb{V}[\![\cdot]\!]$: $\texttt{PVal} \rightarrow Val \cup Var$ and $\mathbb{T}[\![\cdot]\!] : \texttt{PTuple} \rightarrow \mathcal{T}$ can be found in Table 12. The syntactic categories are those of Table 1.

$\mathbb{L}[\![\mathtt{1_s}]\!] = \mathcal{S}$	$\mathbb{L}[\![s]\!] = \{s\}$	$\mathbb{L}[\![l]\!] = \{l\}$	$\mathbb{L}[\![u]\!] = \{u\}$
$\mathbb{V}[\![\mathtt{1_v}]\!] = Val$	$\mathbb{V}[\![v]\!] = \{v\}$	$\mathbb{V}[\![x]\!] = \{x\}$	
$\mathbb{T}[\![\mathtt{1_t}]\!] = \mathcal{T}$	$\mathbb{T}[\![\mathtt{lp}]\!] = \mathbb{L}[\![\mathtt{lp}]\!]$	$\mathbb{T}[\![\mathtt{vp}]\!] = \mathbb{V}[\![\mathtt{vp}]\!]$	$\mathbb{T}[\![\mathtt{pp}]\!] = \mathbb{P}[\![\mathtt{pp}]\!]$
$\mathbb{T}[\![!u]\!] = \{!u\}$	$\mathbb{T}[\![!x]\!] = \{!x\}$	$\mathbb{T}[\![!X]\!] = \{!X\}$	

Table 12: Interpretation for predicates

To define $\mathbb{P}[\![]\!]$ we need to introduce a transition relation for describing possible computations of processes. Relations $\twoheadrightarrow$ and $\rightarrowtail$ introduced in the previous section are not adequate, because they describe the actual computation of nets and processes. The relation we need, instead, has to describe, using a sort of *abstract interpretation*, what actions a process might perform during its computation. Let $\mathcal{V}$ be a set of variables, we will write $P \overset{act}{\underset{\mathcal{V}}{\Longrightarrow}} Q$ if:

- the process P, at some point of its computation, might perform the action act;
- all the actions that are execute before act, do not bind variables in $\mathcal{V}$.

Definition 3.3 *Let $\mathcal{V} \subset VLoc \cup \chi \cup Var$ the relation $\rightarrow_{\mathcal{V}} \subset Proc \times Proc$ is defined as follows:*

- *if act does not bind variables in $\mathcal{V}$ then*
$$act P \rightarrow_{\mathcal{V}} P$$
- *for every P and Q*
$$P|Q \rightarrow_{\mathcal{V}} P$$
$$P|Q \rightarrow_{\mathcal{V}} Q$$
$$P + Q \rightarrow_{\mathcal{V}} P$$
$$P + Q \rightarrow_{\mathcal{V}} Q$$
- *if $A(\widetilde{X}, \widetilde{u}, \widetilde{x}) \overset{def}{=} P$ then*
$$A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle \rightarrow_{\mathcal{V}} P[\widetilde{P}/\widetilde{X}, \widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}]$$

The relation $\overset{}{\underset{\mathcal{V}}{\Longrightarrow}} \subset Proc \times Act \times Proc$ is defined as follows:

- *for every* $\mathcal{V}$,

$$act.P \overset{act}{\underset{\mathcal{V}}{\Longrightarrow}} P$$

- *if* $P \rightarrow_{\mathcal{V}} P'$ *and* $P' \overset{act}{\underset{\mathcal{V}}{\Longrightarrow}} Q$ *then*

$$P \overset{act}{\underset{\mathcal{V}}{\Longrightarrow}} Q$$

Functions $\mathbb{P}[\![\cdot]\!] : \texttt{PProc} \rightarrow \mathit{Proc}$ and $\mathbb{AC}[\![\cdot]\!] : \texttt{PAcc} \rightarrow \mathit{Act}$ are defined as follows:

$\mathbb{P}[\![\mathtt{1_P}]\!] = \mathit{Proc}$

$\mathbb{P}[\![X]\!] = \{X\}$

$\mathbb{P}[\![\mathtt{ap} \rightarrow \mathtt{pp}]\!] = \{P | \exists act, Q, \mathtt{ap}'$
$\mathtt{ap} =_\alpha \mathtt{ap}', act \in \mathbb{P}[\![\mathtt{ap}']\!], Q \in \mathbb{P}[\![\mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]]\!], P \overset{act}{\underset{\mathtt{fv}(\mathtt{ap} \rightarrow \mathtt{pp})}{\Longrightarrow}} Q\}$

$\mathbb{P}[\![\mathtt{pp}_1 \wedge \mathtt{pp}_2]\!] = \mathbb{P}[\![\mathtt{pp}_1]\!] \cap \mathbb{P}[\![\mathtt{pp}_2]\!]$

$\mathbb{AC}[\![\mathtt{o}(\mathtt{tp})@\mathtt{lp}]\!] = \{(\mathbf{out}(t)@\ell | t \in \mathbb{P}[\![\mathtt{tp}]\!] \, , \, \ell \in \mathbb{P}[\![\mathtt{lp}]\!]\}$

$\mathbb{AC}[\![\mathtt{i}(\mathtt{tp})@\mathtt{lp}]\!] = \{(\mathbf{in}(t)@\ell | t \in \mathbb{P}[\![\mathtt{tp}]\!] \, , \, \ell \in \mathbb{P}[\![\mathtt{lp}]\!]\}$

$\mathbb{AC}[\![\mathtt{r}(\mathtt{tp})@\mathtt{lp}]\!] = \{(\mathbf{read}(t)@\ell | t \in \mathbb{P}[\![\mathtt{tp}]\!] \, , \, \ell \in \mathbb{P}[\![\mathtt{lp}]\!]\}$

$\mathbb{AC}[\![\mathtt{e}(\mathtt{pp})@\mathtt{lp}]\!] = \{\mathbf{eval}(Q)@\ell | \ell \in \mathbb{P}[\![\mathtt{lp}]\!], \, Q \in \mathbb{P}[\![\mathtt{pp}]\!]\sigma\}$

$\mathbb{AC}[\![\mathtt{n}(u)]\!] = \{(\mathbf{newloc}(u') | u' \in \mathit{VLoc}\}$

We will write $P : \mathtt{pp}$, $t : \mathtt{tp}$ and $act : \mathtt{ap}$ if $P \in \mathbb{P}[\![\mathtt{pp}]\!]$, $t \in \mathbb{T}[\![\mathtt{tp}]\!]$ and $act \in \mathbb{P}[\![\mathtt{ap}]\!]$ respectively. Conversely, we will write $\neg(P : \mathtt{pp})$, $\neg(t : \mathtt{tp})$ and $\neg(act : \mathtt{ap})$ if $P \notin \mathbb{P}[\![\mathtt{pp}]\!]$, $t \notin \mathbb{P}[\![\mathtt{tp}]\!]$ and $act \notin \mathbb{P}[\![\mathtt{ap}]\!]$. Furthermore, we assume that process predicates are equal up to contraction (i.e. $\mathtt{pp} \wedge \mathtt{pp} = \mathtt{pp}$), commutative and associative properties; for instance $\mathtt{pp}_1 \wedge (\mathtt{pp}_2 \wedge \mathtt{pp}_1) = \mathtt{pp}_1 \wedge \mathtt{pp}_2$.

If we consider predicate $\mathtt{pp} = \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(\mathtt{1_P})@u \rightarrow \mathtt{1_P}$, introduced in the previous Section, we have that:

$$\mathbf{in}(!u_1)@s_1.\mathbf{in}(!x)@u_1.\mathbf{eval}(P)@u_1.Q : \mathtt{pp}$$

indeed:

$$\mathbf{in}(!u_1)@s_1.\mathbf{in}(!x)@u_1.\mathbf{eval}(P)@u_1.Q \overset{\mathbf{in}!u_1@s_1}{\underset{\emptyset}{\Longrightarrow}} \mathbf{in}(!x)@u_1.\mathbf{eval}(P)@u_1.Q \overset{\mathbf{eval}(P)@u_1}{\underset{\{u_1\}}{\Longrightarrow}} Q$$

Moreover:

$\mathbf{in}(!u_1)@s_1 : \mathtt{i}(!u_1)@s_1 =_\alpha \mathtt{i}!u@s_1$,

$\mathtt{e}(\mathtt{1_P})@u[\mathtt{i}(!u_1)@s_1/\mathtt{i}!u@s_1] = \mathtt{e}(\mathtt{1_P})@u_1$,

$\mathbf{eval}(P)@u_1 : \mathtt{e1_P}@u_1$ and $Q : \mathtt{1_P}$,

$\mathbf{in}(!u_1)@s_1.\mathbf{in}(!x)@u_1.\mathbf{eval}(P)@u_1.Q : \mathtt{pp}$

On the contrary, the process

$$\mathbf{in}(!u_2)@s_1.\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.\mathbf{nil}$$

do not satisfies the predicate $\mathtt{pp}$, indeed:

$$\mathbf{in}(!u_2)@s_1.\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.\mathbf{nil} \xrightarrow[\emptyset]{\mathbf{in}(!u_2)@s_1} \mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.\mathbf{nil}$$

Even if action $\mathbf{in}(!u_2)@s_1 : \mathtt{i}(!u_2)@s_1 =_\alpha \mathtt{i}(!u)@s_1$, process $\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.\mathbf{nil}$ does not satisfy $\mathtt{e}(\mathtt{1_P})@u_2 \rightarrow \mathtt{1_P}$. Indeed:

- the action $\mathbf{read}(!u_2)@s_2$ binds the variable u_2;
- there is no process P' such that $\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.\mathbf{nil} \rightarrow_{u_2} P'$.

Hence, $\mathbf{in}(!u_2)@s_1.\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.\mathbf{nil} \rightarrow_{u_2} P'$ does not satisfy $\mathtt{pp}$.

We would like to remark that process predicates represent *set* of *causal dependent* sequences of *accesses* that a single process *might* perform and not actual computational sequences. Thus process predicates are not able to distinguish $P_1|P_2$ from $P_1 + P_2$. Indeed, in $P_1|P_2$, like in $P_1 + P_2$, the accesses that P_1 might perform are not *dependent* on those that P_2 might perform. In that sense, $P_1|P_2$ and $P_1 + P_2$ satisfy the same set of predicates.

3.2.2 A proof system for process and tuple predicates

In this section, we will introduce a proof system that permits verifying the satisfiability of a predicate by a process or a tuple. For the proof system we will prove soundness and completeness.

Definition 3.4 Process, action *and* tuple sequents *are of the form:*

$$\Delta \vdash P : \mathtt{pp} \ , \ \Delta \vdash act : \mathtt{ap} \ , \ \Delta \vdash t : \mathtt{tp}$$

where the assumptions Δ *are sets of pairs* <process variable,process predicate>.

In order to limit the size of the set of predicates that can appear in a proof, we introduce the operators $\downarrow \cdot$ and $\Downarrow \cdot$.

Definition 3.5 *Let* $\mathtt{pp}$ *be a process predicate and let* $\{\widetilde{\mathtt{pp}}\}$ *be a set of process predicates, we define* $\downarrow \mathtt{pp}$ *and* $\Downarrow \{\widetilde{\mathtt{pp}}\}$ *as follows:*

- $\downarrow \mathtt{1_P} = \{\mathtt{1_P}\}$;
- $\downarrow (\mathtt{pp}_1 \wedge \mathtt{pp}_2) = \downarrow \mathtt{pp}_1 \cup \downarrow \mathtt{pp}_2 \cup (\downarrow \mathtt{pp}_1) \wedge (\downarrow \mathtt{pp}_2)$
- $\downarrow (\mathtt{ap} \rightarrow \mathtt{pp}) = \downarrow \mathtt{pp} \cup \downarrow \mathtt{ap} \cup (\mathtt{ap} \rightarrow \downarrow \mathtt{pp})$

and

- $\Downarrow \emptyset = \emptyset$;

Process rules:

$$\Delta \vdash P : \mathtt{1_p} \quad \Delta, X : \mathtt{pp} \vdash X : \mathtt{pp} \quad \Delta \vdash X : X\,(1) \quad \mathtt{Rw}\ \frac{\Delta, X : \mathtt{pp}_i \vdash P : \mathtt{pp}}{\Delta, X : \mathtt{pp}_1 \wedge \mathtt{pp}_2 \vdash P : \mathtt{pp}}\ i \in \{1,2\}$$

$$\mathtt{R}\wedge\ \frac{\Delta \vdash P : \mathtt{pp}_1 \quad \Delta \vdash P : \mathtt{pp}_2}{\Delta \vdash P : \mathtt{pp}_1 \wedge \mathtt{pp}_2} \qquad \mathtt{R}|\ \frac{\Delta \vdash P_i : \mathtt{ap} \to \mathtt{pp}}{\Delta \vdash P_1 | P_2 : \mathtt{ap} \to \mathtt{pp}}\ i \in \{1,2\}$$

$$\mathtt{R}+\ \frac{\Delta \vdash P_i : \mathtt{ap} \to \mathtt{pp}}{\Delta \vdash P_1 + P_2 : \mathtt{ap} \to \mathtt{pp}}\ i \in \{1,2\} \qquad \mathtt{RAct_1}\ \frac{\Delta \vdash P : \mathtt{ap} \to \mathtt{pp}}{\Delta \vdash act.P : \mathtt{ap} \to \mathtt{pp}}\ (2)$$

$$\mathtt{RAct_2}\ \frac{\Delta \vdash act : \mathtt{ap}' \quad \Delta \vdash P : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]}{\Delta \vdash act.P : \mathtt{ap} \to \mathtt{pp}}\ (3)$$

$$\mathtt{RCall}\ \frac{X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash P'[\widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}] : \mathtt{ap} \to \mathtt{pp} \quad \Delta \vdash P_1 : \mathtt{pp}_1 \ \ldots\ \Delta \vdash P_n : \mathtt{pp}_n}{\Delta \vdash A\langle P_1, \ldots, P_n, \widetilde{\ell}, \widetilde{v}\rangle : \mathtt{ap} \to \mathtt{pp}}\ (4)$$

Action rules:

$$\mathtt{ROut}\ \frac{\Delta \vdash t : \mathtt{tp} \quad \Delta \vdash \ell : \mathtt{lp}}{\Delta \vdash \mathbf{out}(t)@\ell : \mathtt{o}(\mathtt{tp})@\mathtt{lp}} \qquad \mathtt{RRead}\ \frac{\Delta \vdash t : \mathtt{tp} \quad \Delta \vdash \ell : \mathtt{lp}}{\Delta \vdash \mathbf{read}(t)@\ell : \mathtt{r}(\mathtt{tp})@\mathtt{lp}}$$

$$\mathtt{RIn}\ \frac{\Delta \vdash t : \mathtt{tp} \quad \Delta \vdash \ell : \mathtt{lp}}{\Delta \vdash \mathbf{in}(t)@\ell : \mathtt{i}(\mathtt{tp})@\mathtt{lp}} \qquad \mathtt{REval}\ \frac{\Delta \vdash P : \mathtt{pp} \quad \Delta \vdash \ell : \mathtt{lp}}{\Delta \vdash \mathbf{eval}(P)@\ell : \mathtt{e}(\mathtt{pp})@\mathtt{lp}}$$

$$\mathtt{RNew}\ \Delta \vdash \mathbf{newloc}(u) : \mathtt{n}(u)$$

Tuple rules:

$$\Delta \vdash !u : !u \quad \Delta \vdash !X : !X \quad \Delta \vdash v : v \quad \Delta \vdash v : \mathtt{1_v} \quad \Delta \vdash \ell : \ell \quad \Delta \vdash \ell : \mathtt{1_s} \quad \Delta \vdash t : \mathtt{1_t}$$

$$\frac{\Delta \vdash t_1 : \mathtt{tp}_1 \quad \Delta \vdash t_2 : \mathtt{tp}_2}{\Delta \vdash t_1, t_2 : \mathtt{tp}_1, \mathtt{tp}_2}\ \mathtt{RTuple}$$

(1) X does not appear in Δ (2) *act* does not bind free variables in $\mathtt{ap} \to \mathtt{pp}$

(3) $\mathtt{ap} =_\alpha \mathtt{ap}'$

(4) $A(X_1, \ldots, X_n, \widetilde{u}, \widetilde{x}) \stackrel{def}{=} P'$ and $\forall\, i.\ \mathtt{pp}_i \in (\downarrow \mathtt{ap} \to \mathtt{pp})^C$

Table 13: The Proof System for process and tuple predicates

- $\Downarrow \{pp\} \cup \{\widetilde{pp}\} = (\downarrow pp) \cup (\Downarrow \{\widetilde{pp}\}) \cup ((\downarrow pp) \wedge (\Downarrow \{\widetilde{pp}\}))$

where:

$$(\downarrow pp_1) \wedge (\downarrow pp_2) = \{pp' \wedge pp'' | pp' \in \downarrow pp_1 \text{ and } pp'' \in \downarrow pp_2\}$$

$$\downarrow ap = \Downarrow \{pp | pp \in ap\}$$

$$(ap \rightarrow \downarrow pp) = \{ap \rightarrow pp' | pp' \in \downarrow pp\}$$

$$\downarrow tp = \Downarrow \{pp | pp \in tp\}$$

Let $\{pp\}$ *be a set of process predicates, we use* $(\{\widetilde{pp}\})^C$ *for the* $\wedge$*-closure of* $\{pp\}$.

The proof system for process and tuples predicates is defined in Table 13. Three groups of rules are considered: *process rules*, *action rules* and *tuple rules*. The more interesting are *process rules* while action and tuple rules are, in some sense, standard.

Process rules contain three axioms which, respectively, state that: every process satisfies $\mathtt{1_P}$; if it is assumed that X satisfies $\mathtt{pp}$ then $X : \mathtt{pp}$ is provable; and every process variable X *satisfies itself*. The condition "X *does not appear in* Δ on axiom $\Delta \vdash X : X$ is used to avoid name clash between formal variables and parameter variables in process definitions. There is a *weakening rule* (Rw): if one is able to prove $P : \mathtt{pp}$ under $\Delta \cup \{X : \mathtt{pp}_1\}$ then $P : \mathtt{pp}$ is provable under $\Delta \cup \{X : \mathtt{pp}_1 \wedge \mathtt{pp}_2\}$. The rule for predicate conjunction (R$\wedge$) ($\mathtt{pp}_1 \wedge \mathtt{pp}_2$) is as usual: $P : \mathtt{pp}_1 \wedge \mathtt{pp}_2$ under Δ if both $P : \mathtt{pp}_1$ and $P : \mathtt{pp}_2$ under Δ. Parallel composition and non deterministic choice are treated in the proof system like *disjunctions*. There exists a proof for $\Delta \vdash P_1 | P_2$ or $\Delta \vdash P_1 + P_2$ if there exists a proof either for $\Delta \vdash P_1 : \mathtt{pp}$ or for $\Delta \vdash P_2 : \mathtt{pp}$ (rules R$|$and R$+$). Two rules can be applied to prove $\Delta \vdash act.P : \mathtt{ap} \rightarrow \mathtt{pp}$ ($\mathtt{RAct_1}$and $\mathtt{RAct_2}$). If act does not bind free variables in $\mathtt{ap} \rightarrow \mathtt{pp}$ and there exists a proof for $\Delta \vdash P : \mathtt{ap} \rightarrow \mathtt{pp}$ then there exists a proof for $\Delta \vdash act.P : \mathtt{ap} \rightarrow \mathtt{pp}$. On the other hand, if there exists a proof for $\Delta \vdash act : \mathtt{ap}'$, $\mathtt{ap}' =_\alpha \mathtt{ap}$ and there exists a proof for $\Delta \vdash P : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]$ then there exists a proof for $\Delta \vdash act.P : \mathtt{ap} \rightarrow \mathtt{pp}$. Finally, one is able to prove $A\langle P_1, \ldots, P_n, \widetilde{\ell}, \widetilde{v}\rangle$, where $A(X_1, \ldots, X_n) \stackrel{def}{=} P'$, if there are $\mathtt{pp}_1, \ldots, \mathtt{pp}_n \in (\downarrow \mathtt{pp})^C$ such that there exist proofs for $X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_x \vdash P'[\widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}]$, $\Delta \vdash P_1 : \mathtt{pp}_1, \ldots, \Delta \vdash P_1 : \mathtt{pp}_1$ (RCall).

Previously, we showed that process $\mathbf{in}(!u_1)@s_1.\mathbf{eval}(P)@u_1.Q$ belongs to the interpretation of $\mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(\mathtt{1_P})@u \rightarrow \mathtt{1_P}$. Using the proof system, we can built, for the sequent $\vdash \mathbf{in}(!u_1)@s_1.\mathbf{eval}(P)@u_1.Q : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(\mathtt{1_P})@u \rightarrow \mathtt{1_P}$, the following proof:

$$\dfrac{\dfrac{\vdash !u_1 : !u_1 \quad \vdash s_1 : s_1}{\vdash \mathbf{in}(!u_1)@s_1 : \mathtt{i}(!u_1)@s_1}\,\mathtt{RIn} \qquad \dfrac{\dfrac{\vdash P : \mathtt{1_P} \quad \vdash u_1 : u_1}{\vdash \mathbf{eval}(P)@u_1 : \mathtt{e}(\mathtt{1_P})@u_1}\,\mathtt{REval} \qquad \vdash Q : \mathtt{1_P}}{\vdash \mathbf{eval}(P)@u_1.Q : \mathtt{e}(\mathtt{1_P})@u_1 \rightarrow \mathtt{1_P}}\,\mathtt{RAct_2}}{\vdash \mathbf{in}(!u_1)@s_1.\mathbf{eval}(P)@u_1.Q : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(\mathtt{1_P})@u \rightarrow \mathtt{1_P}}\,\mathtt{RAct_2}$$

Now we are ready to present soundness and completeness results.

Theorem 3.6 (Soundness) *For every process* P*, action act tuple* t*, predicates* pp*,* ap *and* tp *and* $\Delta = X_1 : pp_1, \ldots, X_n : pp_n$, $\Delta \vdash P : pp$, $\Delta \vdash act : ap$ *and* $\Delta \vdash t : tp$ *are provable then for every processes* $P_1, \ldots, P_n$ *such that, for every* i, $P_i : pp_i$*, then* $P[P_1/X_1, \ldots, P_n/X_n] : pp$, $act[P_1/X_1, \ldots, P_n/X_n] : ap$ *and* $t[P_1/X_1, \ldots, P_n/X_n] : tp$.

Proof: We prove, by induction on the number of steps needed to prove $\Delta \vdash P : \mathtt{pp}$, $\Delta \vdash act : \mathtt{ap}$ and $\Delta \vdash t : \mathtt{tp}$, that if there exists a proof for $\Delta \vdash P : \mathtt{pp}$, $\Delta \vdash act : \mathtt{ap}$ and $\Delta \vdash t : \mathtt{tp}$ then for every processes P_1,..., P_n such that, for every i, $P_i : \mathtt{pp}_i$, then $P[P_1/X_1, \ldots, P_n/X_n] : \mathtt{pp}$, $act[P_1/X_1, \ldots, P_n/X_n] : \mathtt{ap}$ and $t[P_1/X_1, \ldots, P_n/X_n] : \mathtt{tp}$.

Base If $\Delta \vdash P : \mathtt{pp}$, $\Delta \vdash act : \mathtt{ap}$ and $\Delta \vdash t : \mathtt{tp}$ are axioms, i.e. provable in 0 steps, the following cases can be distinguished:

- $\Delta \cup \{X_i : \mathtt{pp}_i\} \vdash X_i : \mathtt{pp}_1$, $\Delta \vdash P : \mathtt{1_P}$ or $\Delta \vdash X : X$ then the statement follows easily, indeed: in the first case, $P_i : \mathtt{pp}_i$ for hypothesis; in the second one for each P_1, ..., P_n, $P[P_1/X_1, \ldots, P_n/X_n]$ satisfies $\mathtt{1_P}$ and in the latter, since X does not appear in Δ, $X[P_1/X_1, \ldots, P_n/X_n] = X$ and $X : X$;
- $\Delta \vdash v : \mathtt{1_v}$, $\Delta \vdash v : v$ $\Delta \vdash !u : !u$, $\Delta \vdash !X : !X$, $\Delta \vdash \ell : \mathtt{1_s}$, $\Delta \vdash \ell : \ell$ and $\Delta \vdash t : \mathtt{1_t}$. For each of these case, the statement follows easily.

Inductive Hypothesis For every $X_1 : \mathtt{pp}_1, \ldots, X_i : \mathtt{pp}_i$, if $X_1 : \mathtt{pp}_1, \ldots, X_i \vdash P : \mathtt{pp}$, $X_1 : \mathtt{pp}_1, \ldots, X_i :\vdash act : \mathtt{ap}$ and $X_1 : \mathtt{pp}_1, \ldots, X_i \vdash t : \mathtt{tp}$ are provable within n steps then for every $P_1 : \mathtt{pp}_1$,...,$P_i : \mathtt{pp}_i$: $P[P_1/X_1, \ldots, P_i/X_i] : \mathtt{pp}$, $act[P_1/X_1, \ldots, P_i/X_i] : \mathtt{ap}$ and $t[P_1/X_1, \ldots, P_i/X_i] : \mathtt{tp}$.

Inductive Step Let us assume that $\Delta \vdash P : \mathtt{pp}$, $\Delta \vdash act : \mathtt{ap}$ and $\Delta \vdash t : \mathtt{tp}$ are provable within $n + 1$ steps. We have to distinguish according to the last applied rule.

(Rw) In that case, the proof starts:

$$\frac{\{X_1 : \mathtt{pp}_1^1, \ldots, X_i : \mathtt{pp}_i\} \vdash P : \mathtt{pp}}{\{X : \mathtt{pp}_1^1 \wedge \mathtt{pp}_1^2, \ldots, X_i : \mathtt{pp}_i\} \vdash P : \mathtt{pp}}$$

Since $\{X_1 : \mathtt{pp}_1^1, \ldots, X_i : \mathtt{pp}_i\} \vdash P : \mathtt{pp}$ is provable within n steps, then:

$$\forall\, P_1 : \mathtt{pp}_1^1, \ldots, P_i : \mathtt{pp}_i \Rightarrow P[P_1/X_1, \ldots, P_i/X_i] : \mathtt{pp}$$

Moreover, for each P if $P : \mathtt{pp}_1 \wedge \mathtt{pp}_2$ then $P : \mathtt{pp}_1$. Hence:

$$\forall\, P_1 : \mathtt{pp}_1^1 \wedge \mathtt{pp}_1^2, \ldots, P_i : \mathtt{pp}_i \Rightarrow P[P_1/X_1, \ldots, P_i/X_i] : \mathtt{pp}$$

(R∧) the statement follows easily from the inductive hypothesis and by definition of $\mathbb{P}[\![\mathtt{pp}_1 \wedge \mathtt{pp}_2]\!] = \mathbb{P}[\![\mathtt{pp}_1]\!] \cap \mathbb{P}[\![\mathtt{pp}_2]\!]$;

(R|and R+) also in that case the statement follows by inductive hypothesis and by noting that, for each P_1 and P_2, if $P_1 : \mathtt{ap} \to \mathtt{pp}$ then both $P_1|P_2$ and $P_1 + P_2$ belong to $\mathbb{P}[\![\mathtt{ap} \to \mathtt{pp}]\!]$;

(RAct$_1$) the proof starts:

$$\frac{\{X_1 : \mathtt{pp}_1, \ldots, X_i : \mathtt{pp}_i\} \vdash P : \mathtt{ap}_1 \to \mathtt{pp}_1}{\{X_1 : \mathtt{pp}_1, \ldots, X_i : \mathtt{pp}_i\} \vdash act.P : \mathtt{ap}_1 \to \mathtt{pp}_1}$$

where *act* does not bind free variable in $\mathtt{ap}_1 \to \mathtt{pp}_1$. From the inductive hypothesis, the statement follows for $\{X_1 : \mathtt{pp}_1, \ldots, X_i : \mathtt{pp}_i\} \vdash P : \mathtt{ap}_1 \to \mathtt{pp}_1$.

Notice that, if $P : \mathtt{ap}_1 \to \mathtt{pp}_1$ then there exist act_1 and P_1 such that: $P \xRightarrow[\mathtt{fv}(\mathtt{ap} \to \mathtt{pp})]{act_1} P_1$ and $act_1 : \mathtt{pp}_1$ and $P_1 : \mathtt{pp}_1$. Moreover, for each *act* which does not bind variables in $\mathtt{fv}(\mathtt{ap} \to \mathtt{pp})$, $act.P \xRightarrow[\mathtt{fv}(\mathtt{ap} \to \mathtt{pp})]{act_1} P_1$, hence $P : \mathtt{ap} \to \mathtt{pp} \Rightarrow act.P : \mathtt{ap} \to \mathtt{pp}$. Since the set variables that are bound by *act* do not change under every substitution, then the statement follows.

(RAct$_2$) Also that case follows by composing inductive hypothesis and definition of $\mathbb{P}[\![\ ]\!]$.

(RCall) In that case the initial step of the actual proof is of the form:

$$\frac{X_1 : \mathtt{pp}_1, \ldots, X_i : \mathtt{pp}_n \vdash \overline{P}[\widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}] : \mathtt{pp} \quad \Delta \vdash P^1 : \mathtt{pp}_1 \ \ldots \ \Delta \vdash P^i : \mathtt{pp}_n}{\Delta \vdash A\langle P^1, \ldots, P^i, \widetilde{\ell}, \widetilde{v}\rangle : \mathtt{pp}}$$

where $\Delta = X'_1 : \mathtt{pp}'_1, \ldots, X'_k : \mathtt{pp}'_k$ and $A(X_1, \ldots, X_i, \widetilde{u}, \widetilde{x}) \stackrel{def}{=} \overline{P}$. From the inductive hypothesis we have that:

- for all $P_1 : \mathtt{pp}_1, \ldots, P_i : \mathtt{pp}_i$ then $\overline{P}[\widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}][P_1/X_1, \ldots, P_i/X_i] : \mathtt{pp}$
- for all $P_1 : \mathtt{pp}'_1, \ldots, P_k : \mathtt{pp}'_k$ then for every $P^1[P_1/X'_1, \ldots, P_k/X'_k] : \mathtt{pp}_1, \ldots, P^i[P_1/X'_1, \ldots, P_k/X'_k] : \mathtt{pp}_i$.

Using these result we have that for every $P_1 : \mathtt{pp}'_1, \ldots, P_k : \mathtt{pp}'_k$:

$$\overline{P}[\widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}][P^1[P'_1/X'_1, \ldots, P'_k/X'_k], \ldots, P^n[P'_1/X'_1, \ldots, P'_k/X'_k]] : \mathtt{pp}$$

then

$$A\langle \widetilde{\ell}, P^1[P'_1/X'_1, \ldots, P'_k/X'_k], \ldots, P^n[P'_1/X'_1, \ldots, P'_k/X'_k], \widetilde{v}\rangle : \mathtt{pp}$$

and

$$A\langle \widetilde{\ell}, P^1, \ldots, P^n, \widetilde{v}\rangle[P'_1/X'_1, \ldots, P'_k/X'_k] : \mathtt{pp}$$

□

Lemma 3.7 *If*

$$\vdash P[P_1/X_1, \ldots P_n/X_n] : \mathtt{pp}$$

$$\vdash act[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap}$$

$$\vdash t[P_1/X_1, \ldots P_n/X_n] : \mathtt{tp}$$

are provable, and $\mathtt{pp} \neq X$, *then there exist* $\mathtt{pp}_1, \ldots, \mathtt{pp}_n$ *such that:*

- *for each i,* $P_i : \mathtt{pp}_i$;
- *for each i,* $\mathtt{pp}_i \in (\downarrow \mathtt{pp})^{\mathsf{C}}$ *(respectively* $\mathtt{pp}_i \in (\downarrow \mathtt{ap})^{\mathsf{C}}$ *and* $\mathtt{pp}_i \in (\downarrow \mathtt{tp})^{\mathsf{C}}$*)*

- *the sequents*

$$X_1 : \mathit{pp}_1, \ldots, X_n : \mathit{pp}_n \vdash P : \mathit{pp}$$

$$X_1 : \mathit{pp}_1, \ldots, X_n : \mathit{pp}_n \vdash \mathit{act} : \mathtt{ap}$$

$$X_1 : \mathit{pp}_1, \ldots, X_n : \mathit{pp}_n \vdash t : \mathtt{tp}$$

  are provable.

Proof: We proceed by induction on the length of the proofs.

Base of Induction If $\vdash P[P_1/X_1, \ldots P_n/X_n] : \mathtt{pp}$ is provable in 1 step then $\mathtt{pp} = \mathtt{1_P}$. Moreover, for each i, we let $\mathtt{pp}_i = \mathtt{1_P}$ and, since $\mathtt{1_P} \in \downarrow \mathtt{pp}^C$, $X_1 : \mathtt{1_P}, \ldots X_n : \mathtt{1_P} \vdash P : \mathtt{1_P}$. Every proof for $\vdash \mathit{act}[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap}$ has a length greatest than 1, while if $\vdash t[P_1/X_1, \ldots P_n/X_n] : \mathtt{tp}$ is provable in one step then either $t = P$ and $\mathtt{tp} = \mathtt{1_P}$, $t = \ell$ and $\mathtt{tp} = \mathtt{lp}$, $t = !\mathit{id}$ and $\mathtt{tp} = !\mathit{id}$ ($\mathit{id} \in \{u, x, X\}$), $t = v$ and $\mathtt{tp} = \mathtt{vp}$ or $\mathtt{tp} = \mathtt{1_t}$. For each of these cases the statement hold.

Inductive Hypothesis If $\vdash P[P_1/X_1, \ldots P_n/X_n] : \mathtt{pp} \vdash \mathit{act}[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap}$ and $\vdash t[P_1/X_1, \ldots P_n/X_n] : \mathtt{tp}$ are provable within m steps then there exist $\mathtt{pp}_1$,...,$\mathtt{pp}_n$ such that:

- for each i, $P_i : \mathtt{pp}_i$;
- for each i, $\mathtt{pp}_i \in (\downarrow \mathtt{pp})^C$, $\mathtt{pp}_i \in (\downarrow \mathtt{ap})^C$ and $\mathtt{pp}_i \in (\downarrow \mathtt{tp})^C$;
- the sequents

$$X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash P : \mathtt{pp}$$

$$X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash \mathit{act} : \mathtt{ap}$$

$$X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash t : \mathtt{tp}$$

  are provable.

Inductive Step The statement follows easily from the inductive hypothesis for act and t_1, t_2. The only interesting cases are those concerning processes. Let $\vdash P[P_1/X_1, \ldots P_n/X_n] : \mathtt{pp}$ be provable in $m + 1$ steps. If $\mathtt{pp} = \mathtt{pp}_1 \wedge \mathtt{pp}_2$ then the statement follows easily by using inductive hypothesis. Let $\mathtt{pp} = \mathtt{ap} \to \mathtt{pp}_1$. We can distinguish three cases according to the last applied rule:

(R|,R+) $P = P^1|P^2$ or $P = P^1 + P^2$ then the proof have to start as follows:

$$\frac{\vdash P^i[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap} \to \mathtt{pp}_1}{\vdash P[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap} \to \mathtt{pp}_1}$$

by using the inductive hypothesis we have that there exists a proof for $X_1 : \mathtt{pp}_1 \ldots X_n : \mathtt{pp}_n \vdash P^i : \mathtt{pp}$. Hence there exists a proof for $X_1 : \mathtt{pp}_1 \ldots X_n : \mathtt{pp}_n \vdash P : \mathtt{pp}$.

($\mathtt{RAct_1}$) $P = \mathit{act}.P^i$ and $\vdash P^i[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap} \to \mathtt{pp}$ is provable within n steps. In that case case we can apply directly the inductive hypothesis.

(RAct$_2$) $P = act.P^i$ and $\vdash act[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap}', \mathtt{ap}' =_\alpha \mathtt{ap}$ and $\vdash P^i[P_1/X_1, \ldots P_n/X_n] : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]$ are provable within n steps. By applying the inductive hypothesis we have that there exists $\mathtt{pp}_1^1, \ldots, \mathtt{pp}_n^1$ and $\mathtt{pp}_1^2, \ldots, \mathtt{pp}_n^2$ such that

$$X_1 : \mathtt{pp}_1^1, \ldots, X_n : \mathtt{pp}_n^1 \vdash act : \mathtt{ap}'$$

$$X_1 : \mathtt{pp}_1^2, \ldots, X_n : \mathtt{pp}_n^2 \vdash P^i : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]$$

Please, notice that, for every i, $\mathtt{pp}_i^1 \wedge \mathtt{pp}_i^2 \in (\downarrow \mathtt{ap}' \to \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}])^C$. The proof, that we are looking for, starts as follows:

$$\dfrac{\dfrac{\dfrac{X_1 : \mathtt{pp}_1^1, \ldots \vdash act : \mathtt{ap}'}{\vdots}}{X_1 : \mathtt{pp}_1^1 \wedge \mathtt{pp}_1^2, \ldots \vdash act : \mathtt{ap}'} \quad \dfrac{\dfrac{X_1 : \mathtt{pp}_1^2, \ldots \vdash P^i : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]}{\vdots}}{X_1 : \mathtt{pp}_1^1 \wedge \mathtt{pp}_1^2, \ldots \vdash P^i : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]}}{X_1 : \mathtt{pp}_1^1 \wedge \mathtt{pp}_1^2, \ldots, X_n : \mathtt{pp}_n^1 \wedge \mathtt{pp}_n^2 \vdash P^i : \mathtt{ap} \to \mathtt{pp}}$$

(RCall) $P = A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle$ then the proof starts as follows:

$$\dfrac{\begin{array}{c} X^1 : \mathtt{pp}^1, \ldots, X^k : \mathtt{pp}^k \vdash P'[\widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}] : \mathtt{ap} \to \mathtt{pp}_1 \\ \vdash P^1[P_1/X_1, \ldots P_n/X_n] : \mathtt{pp}^1 \ldots \vdash P^k[P_1/X_1, \ldots P_n/X_n] : \mathtt{pp}^k \end{array}}{\vdash A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle[P_1/X_1, \ldots P_n/X_n] : \mathtt{ap} \to \mathtt{pp}_1}$$

where $\widetilde{P} = P^1, \ldots, P^k$, $A(X^1, \ldots, X^k, \widetilde{u}, \widetilde{x}) = P'$ and $\mathtt{pp}^1, \ldots, \mathtt{pp}^k \in (\downarrow \mathtt{pp})^C$. By applying inductive hypothesis we have that there exists $\mathtt{pp}_1^1$, …, $\mathtt{pp}_n^1$, …, $\mathtt{pp}_1^k$, …, $\mathtt{pp}_n^k$,such that $X_1 : \mathtt{pp}_1^1, \ldots, X_n : \mathtt{pp}_n^1 \vdash P^1 : \mathtt{pp}^1$, …, $X_1 : \mathtt{pp}_1^k, \ldots, X_n : \mathtt{pp}_n^k \vdash P^k : \mathtt{pp}^k$ are provable, moreover $\mathtt{pp}_i^j \in (\downarrow \mathtt{pp}^j)^C \subseteq (\downarrow \mathtt{pp})^C$. Please notice that for every j $\mathtt{pp}_1^j \wedge \ldots \wedge \mathtt{pp}_n^j$ belongs to $(\downarrow \mathtt{pp})^C$. Moreover, similarly to the previous case, we have that

$$X_1 : \mathtt{pp}_1^1 \wedge \ldots \wedge \mathtt{pp}_1^k, \ldots X_n : \mathtt{pp}_n^1 \wedge \ldots \wedge \mathtt{pp}_n^k \vdash A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle : \mathtt{pp}$$

is provable.

□

Theorem 3.8 (Completeness) *For every P, act, t such that for every $P_1 : \mathtt{pp}_1$, …, $P_n : \mathtt{pp}_n$ implies $P[P_1/X_1, \ldots, P_n/X_n] : \mathtt{pp}$, $act[P_1/X_1, P_n/X_n] : \mathtt{ap}$ $t[P_1/X_1, P_n/X_n] : \mathtt{tp}$ then: $X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash P : \mathtt{pp}$, $X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash act : \mathtt{ap}$ and $X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash t : \mathtt{tp}$ are provable.*

Proof: We proceed by induction on the structure of pp, ap and tp. The only interesting case is $P : \mathtt{ap} \to \mathtt{pp}$, where we assume that for every $\mathtt{ap}'$ and $\mathtt{pp}'$ (subterms of $\mathtt{ap} \to \mathtt{pp}$) if $act' : \mathtt{ap}'$ and $P' : \mathtt{pp}'$ then $\vdash act' : \mathtt{ap}'$ and $\vdash P' : \mathtt{pp}'$ are provable.
If $P : \mathtt{ap} \to \mathtt{pp}$ then there exist $P_1, \ldots, P_n$ such that:

- $P = P_1$;
- $P_i \to_{\mathtt{fv}(\mathtt{ap} \to \mathtt{pp})} P_{i+1}$;
- $P_n = act'.P'$ and $act' : \mathtt{ap}', \mathtt{ap}' =_\alpha \mathtt{ap}$ and $P' : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]$.

We prove that if $P : \mathtt{ap} \to \mathtt{pp}$ then there exists a proof for $\vdash P : \mathtt{ap} \to \mathtt{pp}$ by induction on n.
If $n = 1$ then $P = act'.P'$, $act' : \mathtt{ap}'$, $\mathtt{ap}' =_\alpha \mathtt{ap}$ and $P' : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]$. We have assumed that if $act' : \mathtt{ap}'$ and $P' : \mathtt{pp}'$ then $\vdash act' : \mathtt{ap}'$ and $\vdash P' : \mathtt{pp}'$ are provable, hence $\vdash act.P : \mathtt{ap} \to \mathtt{pp}$ is provable.
We suppose that for every P such that there exist $P_1, \ldots, P_k$ such that:

- $P = P_1$;
- $P_i \to_{\mathtt{fv}(\mathtt{ap} \to \mathtt{pp})} P_{i+1}$;
- $P_k = act'.P'$ and $act' : \mathtt{ap}'$, $\mathtt{ap}' =_\alpha \mathtt{ap}$ and $P' : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]$.

then there exists a proof for $\vdash P : \mathtt{ap} \to \mathtt{pp}$.
Let P be such that $P_1, \ldots, P_k, P_{k+1}$ such that:

- $P = P_1$;
- $P_i \to_{\mathtt{fv}(\mathtt{ap} \to \mathtt{pp})} P_{i+1}$;
- $P_{k+1} = act'.P'$ and $act' : \mathtt{ap}'$, $\mathtt{ap}' =_\alpha \mathtt{ap}$ and $P' : \mathtt{pp}[\mathtt{ap}'/\mathtt{ap}]$.

If $P = P^1|P^2$, $P = P^1 + P^2$ or $P = act_1.P^1$ the thesis follows directly from inductive hypothesis. While if $P = A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{v} \rangle$ and $A(\widetilde{X}, \widetilde{u}, \widetilde{x}) \stackrel{def}{=} P'$ then $P_2 = P'[\widetilde{P}/\widetilde{X}, \widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}]$. For inductive hypothesis we have that there exists a proof for $\vdash P'[\widetilde{P}/\widetilde{X}, \widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}] : \mathtt{ap} \to \mathtt{pp}$. Then, by applying previous lemma, we have that $X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash P'[\widetilde{\ell}/\widetilde{u}, \widetilde{v}/\widetilde{x}]$ is provable. Finally, let $\widetilde{P} = P_1, \ldots, P_n, \mathtt{pp}_1, \ldots, \mathtt{pp}_n \in\downarrow \mathtt{pp}$ and by assumptions we have that $\vdash P_1 : \mathtt{pp}_1, \ldots, \vdash P_n : \mathtt{pp}_n$ are provable. □

We have proved that, for each P and $\mathtt{pp}$, $\vdash P : \mathtt{pp}$ is provable if and only if $P \in \mathbb{P}[\![\mathtt{pp}]\!]$. However, we need also to prove that we are able to decide if there exists a proof for $\vdash P : \mathtt{pp}$.

To obtain this result we first prove that the set of sequents that can be *reached* in a derivation is finite. Then we prove that if there exists a proof for a sequent with *loops* then a proof without loops can be found for the same sequent.

By composing these results we can prove decidability for the proof system in Table 13. Indeed, search a proof for $\Delta \vdash P : \mathtt{pp}$ is equivalent to search paths, in the finite graph induced by the proof system, from $\Delta \vdash P : \mathtt{pp}$ to *axioms*.

Lemma 3.9 *The set of sequents that are reachable in each deduction from $\Delta \vdash P : \mathtt{pp}$ is finite.*

Proof: Let $X_1 : \mathtt{pp}_1, \ldots, X_n : \mathtt{pp}_n \vdash P' : \mathtt{pp}'$ be a sequent reachable in a deduction from $\Delta \vdash P : \mathtt{pp}$ then:

- either $\mathtt{pp}'$ belong to $(\downarrow \mathtt{pp})^C$ or there exists $\mathtt{pp}'' \in (\downarrow \mathtt{pp})^C$ and $\mathtt{pp}'$ is obtained from $\mathtt{pp}''$ by replacing some variables of $\mathtt{pp}''$ with those that appear in P. This follows directly from definition of proof system and from definition of $(\downarrow \mathtt{pp})^C$;
- either $\mathtt{pp}_1, \ldots, \mathtt{pp}_n \in\downarrow \mathtt{pp}$ or $\mathtt{pp}_1, \ldots, \mathtt{pp}_n$ appear in Δ;
- P' is a subterm of P.

Since the set of variables, the set of localities and the set of values that appear in P are finite, then also the set of processes and process predicates that appear in a deduction from $\Delta \vdash P : \mathtt{pp}$ are finite (for every $\mathtt{pp}$ the sets $\downarrow \mathtt{pp}$ and $(\downarrow \mathtt{pp})^C$ are finite). Then the set of sequents that appear in a deduction from $\Delta \vdash P : \mathtt{pp}$ are finite too. □

Lemma 3.10 *If $\Delta \vdash P : \mathtt{pp}$ is provable then there exists a proof without loops.*

Proof: If there exists a proof for $\Delta \vdash P : \mathtt{pp}$ with a loop then the proof would certainly have the following structure:

$$\dfrac{\dfrac{\dfrac{\dfrac{[C]}{\Delta_1 \vdash P_1 : \mathtt{pp}_1}}{\vdots}}{\Delta_1 \vdash P_1 : \mathtt{pp}_1} \quad [B]}{\dfrac{\ddots \qquad\qquad [A]}{\Delta \vdash P : \mathtt{pp}}}$$

This proof can be restructured as follows:

$$\dfrac{\dfrac{[C]}{\Delta_1 \vdash P_1 : \mathtt{pp}_1} \quad [B]}{\dfrac{\ddots \qquad\qquad [A]}{\Delta \vdash P : \mathtt{pp}}}$$

Applying iteratively the procedure we obtain a proof without loops. □

Theorem 3.11 *The existence of a proof for $\Delta \vdash P : \mathtt{pp}$, $\Delta \vdash act : \mathtt{ap}$ and $\Delta \vdash t : \mathtt{tp}$ is decidable.*

Proof: The theorem follows easily from the two previous lemmata. Indeed, Lemma 3.9 establishes that the set of sequents that appear in a proof is finite. Then, the set of proofs without loops is finite too. Moreover, Lemma 3.10 states that if there is a proof for a sequent then there is also a proof without loops. Hence, searching a proof for $\Delta \vdash P : \mathtt{pp}$ corresponds to searching a valid proof in the finite set of proofs without loops. □

Let we consider process $A\langle\rangle$ where

$$A() \stackrel{def}{=} \mathbf{in}(!u_2)@s_1.\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.A\langle\rangle$$

this process does not satisfy predicate: $\mathtt{pp} = \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_\mathtt{P})@u \rightarrow 1_\mathtt{P}$

Indeed, if one looks for proofs without loops for $\vdash A\langle\rangle : \mathtt{pp}$ finds the following uncomplete proofs:

$$\dfrac{\dfrac{\dfrac{\vdash !u_2 : !u_2 \quad \vdash s_1 : s_1}{\vdash \mathbf{in}(!u_2)@s_1 : \mathtt{i}(!u_2)@s_1}\,\mathtt{RIn} \qquad \vdash \mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.A\langle\rangle : \mathtt{e}(1_\mathtt{P})@u_2 \rightarrow 1_\mathtt{P}(1)}{\vdash \mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_\mathtt{P})@u \rightarrow 1_\mathtt{P}}\,\mathtt{RAct}_2}{\vdash A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_\mathtt{P})@u \rightarrow 1_\mathtt{P}}\,\mathtt{RCall}$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_P)@u \rightarrow 1_P(2)}{\vdash \mathbf{eval}(P)@u_2.A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_P)@u \rightarrow 1_P}\,\mathtt{RAct}_1}{\vdash \mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_P)@u \rightarrow 1_P}\,\mathtt{RAct}_1}{\vdash \mathbf{in}(!u_2)@s_1.\mathbf{read}(!u_2)@s_2.\mathbf{eval}(P)@u_2.A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_P)@u \rightarrow 1_P}\,\mathtt{RAct}_1}{\vdash A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_P)@u \rightarrow 1_P}\,\mathtt{RCall}$$

where both the proofs cannot be completed. In the first one, indeed, for the sequent (1) rule $\mathtt{RAct}_1$ cannot be applied, indeed $\mathbf{read}(!u_2)@s_2$ binds variable u_2 that is free in $\mathtt{e}(1_P)@u_2$, while applying rule $\mathtt{RAct}_2$ yields the sequent $\vdash \mathbf{read}(!u_2)@s_2 : \mathtt{e}(1_P)@u_2$, that is not an axiom and for which there are not rule to apply.

In the second proof, indeed, the sequent (2) is the same at the beginning of the proof. The each proof obtained from (2) will contain a loop.

Finally, there are no correct proofs, without loops, for $\vdash A\langle\rangle : \mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_P)@u \rightarrow 1_P$. Hence, $\vdash A\langle\rangle$ does not satisfy $\mathtt{i}(!u)@s_1 \rightarrow \mathtt{e}(1_P)@u \rightarrow 1_P$.

3.3 The proof system for $\mathcal{L}$

We now introduce a proof system for the finite fragment of the logic. The proof rules operate on *sequents* of the form $\vdash N : \phi$. We shall use π to refer to sequents and Π to refer to proofs.

$R1$	$\dfrac{\vdash N : \neg\phi_1 \qquad \vdash N : \neg\phi_2}{\vdash N : \neg(\phi_1 \vee \phi_2)}$	$R2$	$\dfrac{\vdash N : \phi_i}{\vdash N : \phi_1 \vee \phi_2}$
$R3$	$\dfrac{\vdash N' : \phi\{\delta\}}{\vdash N : \langle\mathcal{A}\rangle\phi}\ N \succ\!\!\xrightarrow{a} N', (a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]$	$R4$	$\dfrac{\vdash N : \phi}{\vdash N : \neg\neg\phi}$
$R5$	$\dfrac{\vdash N_1 : \neg\phi\{\delta_1\} \quad \vdash N_2 : \neg\phi\{\delta_2\} \quad \dots}{\vdash N : \neg\langle\mathcal{A}\rangle\phi}\ \left[\forall(a_i;\delta_i) \in \mathbb{A}[\![\mathcal{A}]\!] : \forall N_i \in \left\{N' | N \succ\!\!\xrightarrow{a_i} N'\right\}\right]$		

Table 14: The proof system

The proof system is presented in Table 14. Rules reflect the intuitive semantics of logical operators. Rule $R1$ states that if one proves that N satisfies both $\neg\phi_1$ and $\neg\phi_2$ then N satisfies $\neg(\phi_1 \vee \phi_2)$. Rule $R2$, instead, states that if N satisfies ϕ_1 or ϕ_2 then N satisfies $\phi_1 \vee \phi_2$. The idempotence of $\neg$ is established by rule $R4$. Finally, rules $R3$ and $R5$ are those that regard modal operator $\langle\cdot\rangle$.

Definition 3.12

1. *A sequent* $\vdash N : \phi$ *is* successful *if*

   - $\phi = \mathbf{tt}$;
   - *or* $\phi = \neg\langle\mathcal{A}\rangle\phi'$, *and* $\forall(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!] \nexists N'.N \succ\!\!\xrightarrow{a} N'$;
   - *or* $\phi = \mathtt{tp}@s$ *and there exists* et *such that* $N \xrightarrow{et@s}\!\!\gg N'$ *and* $et : \mathtt{tp}$;
   - *or* $\phi = \neg\mathtt{tp}@s$ *and for all* et *such that* $N \xrightarrow{et@s}\!\!\gg N'$ *then* $\neg(et : \mathtt{tp})$.

2. Π *is a successful proof for* π *if the following conditions hold:*

   - Π *is built using the rules on Table 14;*
   - *and* π *is the root of* Π*;*
   - *and every leaf on* Π *is a successful sequent.*

3.4 Soundness and Completeness

In this section, we prove that for every closed formula ϕ and for every net N if there exists a valid proof for $\vdash N : \phi$ then $N \models \phi$; moreover if $N \models \phi$ then there exists a valid proof for $\vdash N : \phi$.

3.4.1 Soundness

Theorem 3.13 *Let* ϕ *be a formula. If there exists a proof* Π *for* $\vdash N : \phi$ *then* $N \in \mathbb{M}[\![\phi]\!]$.

Proof: The proof goes by induction on the length of Π.

Base of Induction. If length of Π is 0 then it consists of one successful sequent $\vdash N : \phi$. We can have:

- $\phi = \mathbf{tt}$;
- or $\phi = \mathtt{tp}@s$ and there exist et such that $N \xrightarrow{et@s}\!\!\!\!\gg N'$ and $et : \mathtt{tp}$;
- or $\phi = \neg\mathtt{tp}@s$ and for all et such that $N \xrightarrow{et@s}\!\!\!\!\gg N'$ then $\neg(et : \mathtt{tp})$.
- or $\phi = \neg\langle\mathcal{A}\rangle\phi'$ and $\forall(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]$ $\nexists N' : N \succ\!\!\!\xrightarrow{a} N'$.

For each of this cases we have that $N \in \mathbb{M}[\![\phi]\!]$.

Inductive Hypothesis. For every net N and formula ϕ, if $\vdash N : \phi$ is provable in less then n steps then $N \in \mathbb{M}[\![\phi]\!]$.

Inductive Step. Let $\vdash N : \phi$ be provable in $n+1$ steps. We distinguish with respect to last applied rule.

R1 Then $\phi = \neg(\phi_1 \vee \phi_2)$ and the proof starts as follows:

$$\frac{\vdash N : \neg\phi_1 \qquad \vdash N : \neg\phi_2}{\vdash N : \neg(\phi_1 \vee \phi_2)}$$

By using inductive hypothesis we have that $N \in \mathbb{M}[\![\neg\phi_1]\!]$ and $N \in \mathbb{M}[\![\neg\phi_2]\!]$. Then:

$$\begin{array}{rcl} N & \in & (Net - \mathbb{M}[\![\phi_1]\!]) \cap (Net - \mathbb{M}[\![\phi_2]\!]) \\ & = & Net - (\mathbb{M}[\![\phi_1]\!] \cup \mathbb{M}[\![\phi_2]\!]) \\ & = & Net - (\mathbb{M}[\![\phi_1 \vee \phi_2]\!]) \\ & = & \mathbb{M}[\![\neg(\phi_1 \vee \phi_2)]\!] \end{array}$$

R2 Then $\phi = \phi_1 \vee \phi_2$ and the proof starts as follows:

$$\frac{\vdash N : \phi_i}{\vdash N : \phi_1 \vee \phi_2}$$

By using inductive hypothesis we have that $N \in \mathbb{M}[\![\phi_1]\!]$ moreover we have that:

$$\begin{array}{rcl} N & \in & \mathbb{M}[\![\phi_1]\!] \\ & \subseteq & \mathbb{M}[\![\phi_1]\!] \cup \mathbb{M}[\![\phi_2]\!] \\ & = & \mathbb{M}[\![\phi_1 \vee \phi_2]\!] \end{array}$$

R3 Then $\phi = \langle \mathcal{A} \rangle \phi_1$ and the proof is such that:

$$\frac{\vdash N' : \phi_1\{\delta\}}{\vdash N : \langle \mathcal{A} \rangle \phi_1}$$

where there exists $(a; \delta) \in \mathbb{A}[\![\mathcal{A}]\!]$ and N' for which $N \succ\!\!\xrightarrow{a} N'$. By using inductive hypothesis we have that $N' \in \mathbb{M}[\![\phi_1\{\delta\}]\!]$. Moreover by using definition of $\mathbb{M}[\![\ ]\!]$ we have that $N \in \mathbb{M}[\![\langle \mathcal{A} \rangle \phi_1]\!]$.

R4 Then $\phi = \neg\neg\phi_1$ and the proof is such that:

$$\frac{\vdash N : \phi_1}{\vdash N : \neg\neg\phi_1}$$

we have that $N \in \mathbb{M}[\![\phi_1]\!]$ and $\mathbb{M}[\![\neg\neg\phi_1]\!] = \mathbb{M}[\![\phi_1]\!]$.

R5 Then $\phi = \neg\langle \mathcal{A} \rangle \phi'$ and the proof is such that:

$$\frac{\vdash N_1 : \neg\phi'\{\delta_1\} \quad \ldots \quad \vdash N_k : \neg\phi'\{\delta_k\}}{\vdash N : \neg\langle \mathcal{A} \rangle \phi}$$

where

$$\{N_1, \ldots, N_k\} = \bigcup_{(a_i;\delta_i) \in \mathbb{A}[\![\mathcal{A}]\!]} \left\{ N' | N \succ\!\!\xrightarrow{a_i} N' \right\}$$

By using inductive hypothesis we have that

$$\forall (a_i; \delta_i) \in \mathbb{A}[\![\mathcal{A}]\!], \forall N_i \in \left\{ N' | N \succ\!\!\xrightarrow{a_i} N' \right\} : N_i \in \mathbb{M}[\![\neg\phi'\{\delta_i\}]\!]$$

and, for Theorem 2.3, $\left\{ N' | N \succ\!\!\xrightarrow{a} N' \right\}$ is finite. Equivalently

$$\neg \left(\exists (a; \delta) \in \mathbb{A}[\![\mathcal{A}]\!] \exists N' \in \left\{ N' | N \succ\!\!\xrightarrow{a} N' \right\} : N' \in \mathbb{M}[\![\phi'\{\delta\}]\!] \right)$$

then $N \notin \mathbb{M}[\![\langle \mathcal{A} \rangle \phi']\!] \Rightarrow N \in \mathbb{M}[\![\neg\langle \mathcal{A} \rangle \phi']\!]$.

□

3.4.2 Completeness

Lemma 3.14 *Let N be a net, then one between $\vdash N : \phi$ and $\vdash N : \neg\phi$ is provable.*

Proof: The lemma is provable by induction on the syntax of ϕ.

Base of Induction. Let $\phi = \mathtt{tp}@s$ then either $\vdash N : \mathtt{tp}@s$ or $\vdash N : \neg\mathtt{tp}@s$ indeed either $N \xrightarrow{et@s}\!\!\!\!\gg N'$ and $et : \mathtt{tp}$ or for all et such that $N \xrightarrow{et@s}\!\!\!\!\gg N'$ then such that $\neg(et : \mathtt{tp})$.

Inductive Hypothesis. Let ϕ_1 and ϕ_2 be such that either $\vdash N : \phi_i$ or $\vdash N : \neg\phi_i$ are provable ($i = 1, 2$).

Inductive Step.

- if $\phi = \phi_1 \vee \phi_2$ then $\vdash N : \phi$ is provable if and only if $\vdash N : \phi_1$ or $\vdash N : \phi_2$ are provable. For inductive hypothesis either $\vdash N : \phi_i$ or $\vdash N : \neg\phi_i$ are provable. Then if $\vdash N : \phi_1 \vee \phi_2$ is provable then cannot be both $\vdash N : \neg\phi_1$ and $\vdash N : \neg\phi_2$ are provable, then $\vdash N : \neg(\phi_1 \vee \phi_2)$ is not provable.

  On the other hand, if $\vdash N : \neg(\phi_1 \vee \phi_2)$ is provable, then both $\vdash N : \neg\phi_1$ and $\vdash N : \neg\phi_2$ are provable. From the inductive hypothesis we have that both $\vdash N : \phi_1$ and $\vdash N : \phi_2$ are not provable and $\vdash N : \phi_1 \vee \phi_2$ is not provable.

- if $\phi = \langle\mathcal{A}\rangle\phi_1$ and $\vdash N : \langle\mathcal{A}\rangle\phi_1$ is provable if there exists N' such that $N \overset{a}{\succ\!\!\longrightarrow} N'$, $(a;\delta) \in \mathbb{M}[\![\mathcal{A}]\!]$ and $\vdash N'\phi_1\{\delta\}$. By using inductive hypothesis we have that $\vdash N' : \neg\phi_1\{\delta\}$ is not provable hence $\vdash N : \neg\langle\mathcal{A}\rangle\phi_1$ is not provable too. Conversely if $\vdash N : \langle\mathcal{A}\rangle\phi_1$ is not provable then for every N' such that $N \overset{a}{\succ\!\!\longrightarrow} N'$ and $(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!] \vdash N' : \phi_1\{\delta\}$ is not provable. By using inductive hypothesis we have that, for every N', $\vdash N' : \neg\phi_1\{\delta\}$ is provable. We have that the set of N' such that $N \overset{a}{\succ\!\!\longrightarrow} N'$ and $(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!] \vdash N' : \phi_1\{\delta\}$ is finite, then $\vdash N' : \neg\phi$ is provable.

- $\phi = \neg\phi'$, the thesis follows directly from previous cases and by using $\neg\neg\phi = \phi$.

□

Theorem 3.15 *Let N be a net and let ϕ be a closed formula. If $N \in \mathbb{M}[\![\phi]\!]$ then $\vdash N : \phi$ is provable.*

Proof: We suppose that $N \in \mathbb{M}[\![\phi]\!]$ and that $\vdash N : \phi$ is not provable. For the previous lemma we have that $\vdash N : \neg\phi$ is provable, and from the soundness we have also that $N \in \mathbb{M}[\![\neg\phi]\!]$. Thus from the definition of $\mathbb{M}[\![\cdot]\!]$ we have that $N \in Net - \mathbb{M}[\![\phi]\!]$ which is in contradiction with the hypothesis. □

4 Recursive formulae

In this section, we extend the logic with an operator for recursion. We define $\mathcal{L}^*$ as the set of formulae generable from the following grammar, where κ belongs to the set of logical variables $VLog$.

$$\phi ::= \mathbf{tt} \mid \mathtt{tp}@\sigma \mid \langle\mathcal{A}\rangle\phi \mid \kappa \mid \nu\kappa.\phi \mid \phi \vee \phi \mid \neg\phi$$

To guarantee well definedness of the interpretation function of formulae, we shall assume that no variable κ occurs negatively (i.e. under the scope of an odd number of $\neg$ operators) in ϕ.

4.1 Semantics

We have to extend the interpretation function $\mathbb{M}[\![]\!]$ with two new parameters: a substitution environment and a logical environment. This interpretation function follows the schema presented in [16].

Definition 4.1 *We define the* logical environment *Env as*

$$Env \subseteq [VLog \to Subst \to 2^{Net}]$$

We also use ϵ, sometime with indexes, to denote elements of Env.

We define $\mathbb{M}[\![\cdot]\!] : \mathcal{L}^* \to Env \to Subst \to 2^{Net}$ to denote the set of nets that are models of a formula ϕ. The function $\mathbb{M}[\![\cdot]\!]$ is defined as follows:

- $\mathbb{M}[\![\mathbf{tt}]\!]\epsilon\delta = Net$;
- $\mathbb{M}[\![\kappa]\!]\epsilon\delta = \epsilon(\kappa)\delta$;
- $\mathbb{M}[\![\mathtt{tp}@\sigma]\!]\epsilon\delta = \{N | s = \sigma\{\delta\}, \exists et.\ N \xrightarrow{et@s} N', et : \mathtt{tp}\{\delta\}\}$;
- $\mathbb{M}[\![\langle\mathcal{A}\rangle\phi]\!]\epsilon\delta = f_{\mathcal{A}\{\delta\}}(\mathbb{M}[\![\phi]\!]\epsilon)$ where for each $\mathcal{A}$, $f_{\mathcal{A}} : (Subst \to 2^{Net}) \to (Subst \to 2^{Net})$ is defined as follows:

$$f_{\mathcal{A}}(g)\delta = \bigcup_{(a;\delta')\in\mathbb{A}[\![\mathcal{A}]\!]} \{N' | \exists N.\ N' \overset{a}{\rightarrowtail} N\ N \in g\delta' \cdot \delta\}$$

- $\mathbb{M}[\![\phi_1 \vee \phi_2]\!]\epsilon\delta = \mathbb{M}[\![\phi_1]\!]\epsilon\delta \cup \mathbb{M}[\![\phi_2]\!]\epsilon\delta$;
- $\mathbb{M}[\![\neg\phi]\!]\epsilon\delta = Net - \mathbb{M}[\![\phi]\!]\epsilon\delta$;
- $\mathbb{M}[\![\nu\kappa.\phi]\!]\epsilon\delta = \nu f^{\phi}_{\kappa,\epsilon}\delta$ where:
  1. $f^{\phi}_{\kappa,\epsilon} : [Subst \to 2^{Net}] \to [Subst \to 2^{Net}]$ is defined as follows:
  $$f^{\phi}_{\kappa,\epsilon}(g) = \mathbb{M}[\![\phi]\!]\epsilon \cdot [\kappa \mapsto g]$$
  2. $\nu f^{\phi}_{\kappa,\epsilon} = \bigcup\{g | g \subseteq f^{\phi}_{\kappa,\epsilon}(g)\}$ where $g_1 \subseteq g_2$ if and only if for all δ $g_1(\delta) \subseteq g_2(\delta)$.

We will write $N \models \phi$ if $N \in \mathbb{M}[\![\phi]\!]\epsilon_0\delta_0$ where $\epsilon_0 = \lambda\kappa.\emptyset$ and $\delta_0 = []$.

Notice that introducing substitution like explicit parameter in definition of the interpretation function, and assuming that all logical variable appear in formulae positively, permit defining function $\mathbb{M}[\![]\!]$ like composition of monotone functions in $((Subst \to 2^{Net}) \to (Subst \to 2^{Net}), \subseteq)$, where $g_1 \subseteq g_2$ ($g_1, g_2 \in Subst \to 2^{Net}$) if for each δ $g_1(\delta) \subseteq g_2(\delta)$. Moreover, since $\langle(Subst \to 2^{Net}) \to (Subst \to 2^{Net}), \subseteq\rangle$ is a complete lattice, we can use Tarski's Fixed Point Theorem that guarantee existence of a unique *maximal* fixed point for $\mathbb{M}[\![\phi]\!]\epsilon$, whenever logical variables that appear in ϕ are all positive.

4.2 The proof system

We now introduce a tableau based proof system for the $\mathcal{L}^*$ formulae. A tableau based proof system for μ-calculus has been introduced in [6]. Sequents that are used in tableau proofs maintain information about the systems involved in *recursive* properties. This permits avoiding proofs with *loops*.

Sequents presented in Section 3.3 are extended with set of hypothesis H, i.e. a set of pairs $N : \phi$. Hence new rules operate on *sequents* of the form $H \vdash N : \phi$. The proof system is defined in Table 15. Rules **R1-R5** are essentially the same as in Table 14, where we have the rules for the finitary logics. The only difference is the addition of the hypothesis H. Rules **R6** and **R7** are those introduced for handling recursion. There we use the *sub-term* relation ($\prec$) and the *structural congruence* ($\equiv$) which are defined below.

Definition 4.2 *Let ϕ_1 and ϕ_2 be formulae, we say that ϕ_1 is an* immediate sub-term *of ϕ_2, written $\phi_1 \prec_I \phi_2$, if one of the following holds:*

1. $\phi_2 = \neg\phi_1$;
2. $\phi_2 = \phi_1 \vee \phi_3$ *or* $\phi = \phi_3 \vee \phi_1$, *for some* ϕ_3;
3. $\phi_2 = \langle\mathcal{A}\rangle\phi_1$;
4. $\phi_2 = \nu\kappa.\phi_1$.

We use $\prec$ for the transitive closure of $\prec_I$, and $\preceq$ for the transitive and reflexive closure of $\prec_I$.

Definition 4.3 *We define $\equiv$ as the least congruence defined as follows:*

1. $N \equiv N$;
2. $N_1 \parallel N_2 \equiv N_2 \parallel N_1$;
3. $(N_1 \parallel N_2) \parallel N_3 \equiv N_1 \parallel (N_2 \parallel N_3)$;
4. $s ::_\rho P_1 \parallel s ::_\rho P_2 \equiv s ::_\rho P_1|P_2$;
5. $s ::_\rho P|\mathbf{nil} \equiv s ::_\rho P$.

Let $\mathcal{N}$ be a set of KLAIM nets, we will sometime use $\mathcal{N}/\equiv$ like the set $\mathcal{N}$ quotiented with respect to the equivalence $\equiv$.

Like for formulae without recursion, we need to define successful sequents and proofs.

Definition 4.4

1. *A sequent $H \vdash N : \phi$ is* successful *if*
   - $\phi = \mathbf{tt}$;
   - *or* $\phi = \nu\kappa.\phi'$ *and* $\exists N'.N' \equiv N$ *and* $N' : \nu\kappa.\phi' \in H$;
   - *or* $\phi = \neg\langle\mathcal{A}\rangle\phi'$, *and* $\forall(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]\ \nexists N'.N \succ\!\!\xrightarrow{a} N'$;
   - *or* $\phi = \mathtt{tp}@s$ *and there exists et such that* $N \xrightarrow{et@s}\!\!\!\!\rightarrow N'$ *and* $et : \mathtt{tp}$;

$$R1 \quad \frac{H \vdash N : \neg\phi_1 \qquad H \vdash N : \neg\phi_2}{H \vdash N : \neg(\phi_1 \vee \phi_2)} \qquad R2 \quad \frac{H \vdash N : \phi_i}{H \vdash N : \phi_1 \vee \phi_2}$$

$$R3 \quad \frac{H \vdash N' : \phi\{\delta\}}{H \vdash N : \langle\mathcal{A}\rangle\phi} \; N \succ\!\!\xrightarrow{a} N', (a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!] \qquad R4 \quad \frac{H \vdash N : \phi}{H \vdash N : \neg\neg\phi}$$

$$R5 \quad \frac{H \vdash N_1 : \neg\phi\{\delta_1\} \quad H \vdash N_2 : \neg\phi\{\delta_2\} \quad \ldots}{H \vdash N : \neg\langle\mathcal{A}\rangle\phi} \left[\forall (a_i;\delta_i) \in \mathbb{A}[\![\mathcal{A}]\!] : \forall N_i \in \left\{N' | N \succ\!\!\xrightarrow{a_i} N'\right\}\right]$$

$$R6 \quad \frac{H' \cup \{N : \nu\kappa.\phi\} \vdash N : \neg\phi[\nu\kappa.\phi/\kappa]}{H \vdash N : \neg\nu\kappa.\phi} \; [N' \equiv N, N' : \nu\kappa.\phi \notin H]$$

$$R7 \quad \frac{H' \cup \{N' : \nu\kappa.\phi\} \vdash N : \phi[\nu\kappa.\phi/\kappa]}{H \vdash N : \nu\kappa.\phi} \; [N' \equiv N, N' : \nu\kappa.\phi \notin H]$$

where $H' = H - \{N' : \phi' | \nu\kappa.\phi \prec \phi'\}$

Table 15: The proof system

- *or* $\phi = \neg \mathtt{tp}@s$ *and for all et such that* $N \xrightarrow{et@s}\!\!\!\!\gg N'$ *then* $\neg(et : \mathtt{tp})$.

2. Π *is a successful proof for* π *if the following conditions hold:*

   - Π *is built using the rules on Table 15;*
   - *and* π *is the root of* Π*;*
   - *and every leaf on* Π *is a successful sequent.*

4.3 Soundness and Completeness

The proof of soundness and completeness follows the same scheme of section 3.4. However, in order to prove recursive properties for nets we could have to consider an infinite number of sequents. What we shall go to prove is that for every formula ϕ:

- if there exists a successful proof for $\emptyset \vdash N : \phi$ then $N \models \phi$;
- if $N \models \phi$ and the set $\mathcal{N} = \{N' | N \succ\!\!\longrightarrow^* N'\}/\equiv$ is finite then there exists a successful proof for $\emptyset \vdash N : \phi$;
- if $N \models \phi$ and ϕ satisfies some syntactic constraints then there exists a successful proof for $\emptyset \vdash N : \phi$.

First we have to extend definition of $\mathbb{M}[\![\ ]\!]$ in order to consider the set of hypothesis H. We define $\mathbb{M}[\![\phi]\!]^H$ as follows:

- $\mathbb{M}[\![\mathbf{tt}]\!]^H \epsilon\delta = Net$;
- $\mathbb{M}[\![\kappa]\!]^H \epsilon\delta = e(\kappa)\delta$
- $\mathbb{M}[\![\mathtt{tp}@\sigma]\!]^H \epsilon\delta = \{N | s = \sigma\{\delta\}, \exists et.\ N \xrightarrow{et@s}\!\!\!\!\gg N', et : \mathtt{tp}\{\delta\}\}$;
- $\mathbb{M}[\![\langle\mathcal{A}\rangle\phi]\!]^H \epsilon\delta = \{N | \exists a, \delta', N' : N \succ\!\!\xrightarrow{a} N', (a;\delta') \in \mathbb{A}[\![\mathcal{A}\{\delta\}]\!], N' \in \mathbb{M}[\![\phi]\!]^H \epsilon\delta \cdot \delta'\}$;
- $\mathbb{M}[\![\phi_1 \vee \phi_2]\!]^H \epsilon\delta = \mathbb{M}[\![\phi_1]\!]^H \epsilon\delta \cup \mathbb{M}[\![\phi_2]\!]^H \epsilon\delta$;

- $\mathbb{M}[\![\neg\phi]\!]^H\epsilon\delta = Net - \mathbb{M}[\![\phi]\!]^H\epsilon\delta$;
- $\mathbb{M}[\![\nu\kappa.\phi]\!]^H\epsilon\delta = \nu f^{\phi,h}_{\kappa,\epsilon}\delta \cup h\delta$ where:
  1. $f^{\phi,h}_{\kappa,\epsilon} : [Subst \to 2^{Net}] \to [Subst \to 2^{Net}]$ is defined as follows:
  $$f^{\phi,h}_{\kappa,\epsilon}(g) = f^{\phi}_{\kappa,\epsilon}(g \cup h)$$
  2. $h : Subst \to Nets^*$ is defined as follows:
  $$h\delta = \{N | N : \nu\kappa.\phi\{\delta\} \in H\}$$
  3. $\nu f^{\phi,h}_{\kappa,\epsilon} = \bigcup\{g | g \subseteq f^{\phi,h}_{\kappa,\epsilon}(g)\}$.

Please notice that if $H = \emptyset$ then $\mathbb{M}[\![\phi]\!]^H\epsilon\delta = \mathbb{M}[\![\phi]\!]\epsilon\delta$.

We introduce now two technical lemmata. The first one guarantees that the set of hypothesis H does not alter interpretation of formulae. The second one, instead, states that two structural equivalent nets are models for the same set of formulae.

Lemma 4.5 *For $f^{\phi,h}_{\kappa,\epsilon}$ the following properties hold:*

- *$f^{\phi,h}_{\kappa,\epsilon}$ is continuous in $((Subst \to 2^{Net}) \to (Subst \to 2^{Net}), \subseteq)$;*
- *let $h = h_1 \cup h_2$, $h_1 \subseteq \nu f^{\phi,h}_{\kappa,\epsilon}$ if and only if $h_1 \subseteq \nu f^{\phi,h_2}_{\kappa,\epsilon}$;*
- *if $h_1 \subseteq \nu f^{\phi,h_2}_{\kappa,\epsilon}$ then $\nu f^{\phi,h_2}_{\kappa,\epsilon} = \nu f^{\phi,h}_{\kappa,\epsilon}$.*

Proof:

1. This derives directly from the continuity of $f^{\phi}_{\kappa,\epsilon}$.
2. ($\Rightarrow$) $h_1 \subseteq \nu f^{\phi,h}_{\kappa,\epsilon}$ if and only if there exists g such that:
   $$h_1 \subseteq g \subseteq f^{\phi,h}_{\kappa,\epsilon}(g) = f^{\phi}_{\kappa,\epsilon}(g \cup h)$$
   if $h_1 \subseteq g$ then $g \cup h = g \cup h_2$ and
   $$g \subseteq f^{\phi}_{\kappa,\epsilon}(g \cup h_2) = f^{\phi,h_2}_{\kappa,\epsilon}(g)$$
   then $h_1 \subseteq g \subseteq \nu f^{\phi,h_2}_{\kappa,\epsilon}$.
   ($\Leftarrow$) If $h_1 \subseteq \nu f^{\phi,h_2}_{\kappa,\epsilon}$ then there exists g such that
   $$\begin{array}{rcl} h_1 \subseteq g \subseteq f^{\phi,h_2}_{\kappa,\epsilon}(g) & = & f^{\phi}_{\kappa,\epsilon}(g \cup h_2) \\ & = & f^{\phi}_{\kappa,\epsilon}(g \cup h) \\ & = & f^{\phi,h}_{\kappa,\epsilon}(g) \\ & \Rightarrow & g \subseteq f^{\phi,h}_{\kappa,\epsilon}(g) \end{array}$$

3. Let $h_1 \subseteq \nu f_{\kappa,\epsilon}^{\phi,h_2}$. We have that:

$$f_{\kappa,\epsilon}^{\phi,h_2}(\nu f_{\kappa,\epsilon}^{\phi,h_2}) = \nu f_{\kappa,\epsilon}^{\phi,h_2} \tag{1}$$

$$\nu f_{\kappa,\epsilon}^{\phi,h_2} = \nu f_{\kappa,\epsilon}^{\phi,h_2} \cup h_1 \tag{2}$$

then from 1 and 2 we have that

$$\nu f_{\kappa,\epsilon}^{\phi,h_2} = f_{\kappa,\epsilon}^{\phi,h_2}(\nu f_{\kappa,\epsilon}^{\phi,h_2} \cup h_1) = f_{\kappa,\epsilon}^{\phi,h}(\nu f_{\kappa,\epsilon}^{\phi,h_2})$$

then $\nu f_{\kappa,\epsilon}^{\phi,h} \subseteq \nu f_{\kappa,\epsilon}^{\phi,h_2}$. On the other hand we have that

$$\begin{array}{rcl} \nu f_{\kappa,\epsilon}^{\phi,h} & = & f_{\kappa,\epsilon}^{\phi,h}(\nu f_{\kappa,\epsilon}^{\phi,h}) \\ & = & f_{\kappa,\epsilon}^{\phi}(\nu f_{\kappa,\epsilon}^{\phi,h} \cup h) \\ & = & f_{\kappa,\epsilon}^{\phi}(\nu f_{\kappa,\epsilon}^{\phi,h} \cup h_2) \\ & = & f_{\kappa,\epsilon}^{\phi,h_2}(\nu f_{\kappa,\epsilon}^{\phi,h}) \end{array}$$

then $\nu f_{\kappa,\epsilon}^{\phi,h_2} \subseteq \nu f_{\kappa,\epsilon}^{\phi,h}$ and $\nu f_{\kappa,\epsilon}^{\phi,h_2} \subseteq \nu f_{\kappa,\epsilon}^{\phi,h}$.

□

Lemma 4.6 *Let N_1 and N_2 be such that $N_1 \equiv N_2$, then for every formula ϕ, $N_1 \in \mathbb{M}[\![\phi]\!] \Leftrightarrow N_2 \in \mathbb{M}[\![\phi]\!]$.*

Proof: The proof follows directly by induction on the structure of ϕ and by observing that, since $\equiv$ does not involve processes and tuples argument of actions, if $N_1 \equiv N_2$ then $N_1 \stackrel{a}{\rightarrowtail} N_1' \Leftrightarrow N_2 \stackrel{a}{\rightarrowtail} N_2'$ and $N_1' \equiv N_2'$. □

4.3.1 Soundness

Soundness result can be proved analogously to 3.4.1. However, in order to treat difficulties related to recursive function we have to introduce the following lemma.

Lemma 4.7 *For every formula ϕ, set of hypothesis H and closed formula ϕ' such that no strict subformula of $\phi'\{\delta\}$ is in H holds that:*

$$\mathbb{M}[\![\phi[\phi'/\kappa]]\!]^H \epsilon\delta = f_{\kappa,\epsilon}^{\phi}(\mathbb{M}[\![\phi']\!]^H \epsilon)\delta$$

Proof: The lemma is provable by induction on the structure of the formula ϕ.

Base of Induction. If $\phi = \kappa$, $\phi = \texttt{tp@}s$ or $\phi = \neg\texttt{tp@}s$ then the thesis follows easily.

Inductive Hypothesis. Let ϕ_1 and ϕ_2 be such that

$$\mathbb{M}[\![\phi_i[\phi/\kappa]]\!]^H \epsilon\delta = f_{\kappa,\epsilon}^{\phi_i}(\mathbb{M}[\![\phi']\!]\epsilon)\delta$$

Inductive Step.

- $\phi = \phi_1 \vee \phi_2$:

$$\begin{array}{rcl}\mathbb{M}[\![\phi[\phi'/\kappa]]\!]^H\epsilon\delta & = & \mathbb{M}[\![\phi_1[\phi'/\kappa] \vee \phi_2[\phi'/\kappa]]\!]\epsilon\delta \\ & = & \mathbb{M}[\![\phi_1[\phi'/\kappa]]\!]\epsilon\delta \cup \mathbb{M}[\![\phi_2[\phi'/\kappa]]\!]\epsilon\delta \\ & = & f^{\phi_1}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]\epsilon)\delta \cup f^{\phi_2}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]\epsilon)\delta \\ & = & f^{\phi_1 \vee \phi_2}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]\epsilon)\delta\end{array}$$

- $\phi = \nu\kappa_1\phi_1$:

$$\mathbb{M}[\![\phi[\phi'/\kappa]]\!]^H\epsilon\delta = \nu f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon} \cup h\delta$$

where $h\delta = \{N | N : \nu\kappa_1.\phi_1[\phi'/\kappa]\{\delta\} \in H\}$. For hypothesis no formula that involves $\phi'\{\delta\}$ as strict subformula is in H, then, $h\delta = \emptyset$, then

$$\begin{array}{rcl}\nu f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon} \cup h\delta & = & \nu f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon} \\ & = & f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon}(\nu f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon}) \\ & = & \mathbb{M}[\![\phi_1[\phi'/\kappa]]\!]^H e[\kappa_1 \mapsto \nu f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon} \\ & = & \mathbb{M}[\![\phi_1]\!]^H\epsilon[\kappa_1 \mapsto \nu f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon}][\kappa \mapsto \mathbb{M}[\![\phi']\!]^H\epsilon[\kappa_1 \mapsto \nu f^{\phi_1[\phi'/\kappa],h}_{\kappa_1,\epsilon}]]\delta \\ & = & \mathbb{M}[\![\phi_1]\!]^H\epsilon[\kappa \mapsto \mathbb{M}[\![\phi']\!]^H\epsilon][\kappa_1 \mapsto \nu f^{\phi_1,h}_{e[\kappa \mapsto \mathbb{M}[\![\phi']\!]^H\epsilon],\kappa_1}]\delta \\ & = & \mathbb{M}[\![\nu\kappa_1.\phi_1]\!]^H\epsilon[\kappa \mapsto \mathbb{M}[\![\phi']\!]^H\epsilon]\delta \\ & = & f^{\nu\kappa_1.\phi_1}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]\epsilon)\delta\end{array}$$

- $\phi = \langle\mathcal{A}\rangle\phi_1$:

$$\begin{array}{rcl}\mathbb{M}[\![\phi[\phi'/\kappa]]\!]^H & = & f_{\mathcal{A}}(\mathbb{M}[\![\phi_1[\phi'/\kappa]]\!]^H\epsilon)\delta \\ & = & f_{\mathcal{A}}(f^{\phi_1}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]e))\delta \\ & = & f^{\langle\mathcal{A}\rangle\phi_1}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]e)\delta\end{array}$$

- $\phi = \neg\phi_1$:

$$\begin{array}{rcl}\mathbb{M}[\![\phi[\phi'/\kappa]]\!]^H\epsilon\delta & = & Net - \mathbb{M}[\![\phi_1[\phi'/\kappa]]\!]^H\epsilon\delta \\ & = & Net - f^{\phi_1}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]e)\delta \\ & = & f^{\neg\phi_1}_{\kappa,\epsilon}(\mathbb{M}[\![\phi']\!]e)\delta\end{array}$$

□

Theorem 4.8 *If there exists a proof* Π *for* $H \vdash N : \phi$ *then* $N \in \mathbb{M}[\![\phi]\!]^H\epsilon_0\delta_0$.

Proof: The theorem can be proved by induction on the length of Π. The proof follows the same schema of the proof for Theorem 3.13.

Base of Induction. If length of Π is 0 then we can have:

- $\phi = \mathbf{tt}$.
- $\phi = \nu\kappa.\phi'$ and $\exists N'.N' : \nu\kappa.\phi' \in H$ and $N' \equiv N$;
- $\phi = \neg\langle\mathcal{A}\rangle\phi'$, and $\forall(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!] \not\exists N'.N \succ\!\!\!\xrightarrow{a} N'$;

- $\phi = \mathtt{tp}@s$ and there exists et such that $N \xrightarrow{et@s} N'$ and $et : \mathtt{tp}$;
- $\phi = \neg\mathtt{tp}@s$ and for all et such that $N \xrightarrow{et@s} N'$ then $\neg(et : \mathtt{tp})$.

For each of these cases we have that $N \in \mathbb{M}[\![\phi]\!]^H$.

Inductive Hypothesis. We suppose that if $H \vdash N : \phi$ is provable in less then n steps then $N \in \mathbb{M}[\![\phi]\!]^H$.

Inductive Step. Let $H \vdash N : \phi$ provable in $n+1$ steps. We distinguish with respect to last applied rule. All cases are a generalization of the Theorem 3.13, the only new cases are $R6$ and $R5$. We have to prove that

$$\mathbb{M}[\![\nu\kappa.\phi]\!]^H \epsilon\delta = \mathbb{M}[\![\phi[\nu\kappa.\phi/\kappa]]\!]^{H \cup N:\nu\kappa.\phi\{\delta\}} \epsilon\delta$$

and no formulae in H involve $\nu\kappa.\phi$ as strict subformula and $N : \nu\kappa.\phi \notin H$.
For Lemma 4.7 we have that

$$\mathbb{M}[\![\phi[\nu\kappa.\phi/\kappa]]\!]^{H \cup N:\nu\kappa.\phi\{\delta\}} e\delta = f^{\phi,h}_{\kappa,\epsilon}\left(\mathbb{M}[\![\nu\kappa.\phi]\!]^{H \cup N:\nu\kappa.\phi\{\delta\}}\epsilon\right)$$

where $h\delta' = \{N' | N' : \nu\kappa.\phi\delta' \in H \cup \{N : \nu\kappa.\phi\{\delta\}\}\}$. From definition of $\mathbb{M}[\![\cdot]\!]^H$ we have that

$$\mathbb{M}[\![\nu\kappa.\phi]\!]^{H \cup N:\nu\kappa.\phi}\epsilon = \nu f^{\phi,h}_{\kappa,\epsilon} \cup h$$

then we have

$$\mathbb{M}[\![\phi[\nu\kappa.\phi/\kappa]]\!]^{H \cup N:\nu\kappa.\phi\{\delta\}} \epsilon\delta = f^{\phi,h}_{\kappa,\epsilon}(\nu f^{\phi,h}_{\kappa,\epsilon} \cup h) = \nu f^{\phi,h}_{\kappa,\epsilon}\delta$$

From inductive hypothesis we have that $N \in \nu f^{\phi,h}_{\kappa,\epsilon}\delta$ then

$$h' = \lambda\delta'.\{N | \nu\kappa.\phi\{\delta\} = \nu\kappa.\phi\{\delta'\}\} \subseteq \nu f^{\phi,h}_{\kappa,\epsilon}$$

and $h = h_1 \cup h'$, where $h_1\delta' = \{N | N : \phi\{\delta'\} \in H\}$. For previous lemma we have that

$$\nu f^{\phi,h}_{\kappa,\epsilon}\delta = \nu f^{\phi,h_1}_{\kappa,\epsilon} = \mathbb{M}[\![\nu\kappa.\phi]\!]^H\epsilon\delta$$

then the thesis follows. □

4.3.2 Finiteness of Proofs

We want now prove that if N is such that the set of nets that are reachable form N, quotiented with respect to relation $\equiv$, is finite then the sequents $H \vdash N : \phi$, for every formula ϕ and hypothesis H, has only finite proofs.

First, we introduce some technical tools. Given two sequents π_1 and π_2 we define $\pi_1 \sqsubseteq_I \pi_2$ as the least relation satisfying:

- $H \vdash N : \phi_1 \vee \phi_2 \sqsubseteq_I H \vdash N : \phi_i$;
- $H \vdash N : \langle\mathcal{A}\rangle\phi \sqsubseteq_I H \vdash N' : \phi'$, where $N \succ\!\!\!\xrightarrow{a} N'$, $(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]$ and $\phi' = \phi\{\delta\}$;
- $H \vdash N : \phi_1 \vee \phi_2 \sqsubseteq_I H \vdash N : \phi_i$;

- if $N : \nu\kappa.\phi \notin H$ then $H \vdash N : \nu\kappa.\phi \sqsubseteq_I H' \cup N : \nu\kappa.\phi \vdash N : \phi[\nu\kappa.\phi/\kappa]$, where

$$H' = H - \{N' : \phi' | \nu\kappa.\phi \prec \phi'\}$$

- if $H_1 \vdash N_1 : \phi_1 \sqsubseteq_I H_2 \vdash N_2 : \phi_2$ then $H_1 \vdash N_1 : \neg\phi_1 \sqsubseteq_I H_2 \vdash N_2 : \neg\phi_2$.

We also define $\sqsubseteq$ as the transitive closure of $\sqsubseteq_I$.

We denote with $\{\pi_i\}_{i\in I}$, where $I \subset \mathbb{N}$, a sequence of sequents such that, for all $i, j \in I$, if $i < j$ then $\pi_i \sqsubseteq \pi_j$, we also use H_i, N_i and ϕ_i for set of hypothesis, net and formula in π_i ($\pi_i = H_i \vdash N : \phi_i$).

Definition 4.9 *Let $\mathcal{N}$ be a set of nets, we define $\mathcal{A}(\mathcal{N})$ as follows:*

$$\mathcal{A}(\mathcal{N}) = \{a | \exists N_1, N_2 \in \mathcal{N} : \ N_1 \overset{a}{\rightarrowtail} N_2\}$$

Definition 4.10 *Let $\mathcal{N}$ be a set of nets we define $\mathcal{R}_{\mathcal{N}}(\phi)$ as follows:*

- $\mathcal{R}_{\mathcal{N}}(\mathtt{tp}@s) = \{\mathtt{tp}@s\}$;
- $\mathcal{R}_{\mathcal{N}}(\neg\mathtt{tp}@s) = \{\neg\mathtt{tp}@s\}$;
- $\mathcal{R}_{\mathcal{N}}(\kappa) = \emptyset$;
- $\mathcal{R}_{\mathcal{N}}(\neg\neg\phi) = \mathcal{R}_{\mathcal{N}}(\phi) \cup \{\neg\neg\phi\}$;
- $\mathcal{R}_{\mathcal{N}}(\langle\mathcal{A}\rangle\phi) = \bigcup_{a\in\mathcal{A}(\mathcal{N})} \bigcup_{(a;\delta)\in\mathbb{A}[\![\mathcal{A}]\!]} \mathcal{R}_{\mathcal{N}}(\phi\{\delta\}) \cup \{\langle\mathcal{A}\rangle\phi\}$
- $\mathcal{R}_{\mathcal{N}}(\nu\kappa.\phi) = \mathcal{R}_{\mathcal{N}}(\phi)[\nu\kappa.\phi/\kappa] \cup \{\nu\kappa.\phi\}$;
- $\mathcal{R}_{\mathcal{N}}(\neg\nu\kappa.\phi) = \mathcal{R}_{\mathcal{N}}(\neg\phi)[\nu\kappa.\phi/\kappa] \cup \{\neg\nu\kappa.\phi\}$;
- $\mathcal{R}_{\mathcal{N}}(\phi_1 \vee \phi_2) = \mathcal{R}_{\mathcal{N}}(\phi_1) \cup \mathcal{R}_{\mathcal{N}}(\phi_2) \cup \{\phi_1 \cup \phi_2\}$;
- $\mathcal{R}_{\mathcal{N}}(\neg(\phi_1 \vee \phi_2)) = \mathcal{R}_{\mathcal{N}}(\neg\phi_1) \cup \mathcal{R}_{\mathcal{N}}(\neg\phi_2) \cup \{\neg(\phi_1 \cup \phi_2)\}$;

Lemma 4.11 *For every finite set of nets $\mathcal{N}$ and for every formula ϕ $\mathcal{R}_{\mathcal{N}}(\phi)$ is finite.*

Proof: We prove the lemma by induction on the structure of ϕ.

Base of Induction. If $\phi = \mathtt{tp}@s$, $\phi = \neg\mathtt{tp}@s$ or $\phi = \kappa$ then $\mathcal{R}_{\mathcal{N}}(\phi)$ is finite.

Inductive Hypothesis. We suppose that for every formulae ϕ_1 and ϕ_2 if $\mathcal{N}$ is finite then $\mathcal{R}_{\mathcal{N}}(\phi_1)$ and $\mathcal{R}_{\mathcal{N}}(\phi_2)$ are finite too.

Inductive Step.

- $\phi = \phi_1 \vee \phi_2$, by definition of $\mathcal{R}_{\mathcal{N}}$ we have that

$$\mathcal{R}_{\mathcal{N}}(\phi_1 \vee \phi_2) = \{\phi_1 \vee \phi_2)\} \cup \mathcal{R}_{\mathcal{N}}(\phi_1) \cup \mathcal{R}_{\mathcal{N}}(\phi_2)$$

  from the inductive hypothesis we have that $\mathcal{R}_{\mathcal{N}}(\phi_1)$ and $\mathcal{R}_{\mathcal{N}}(\phi_2)$ are finite, then $\mathcal{R}_{\mathcal{N}}(\phi_1 \vee \phi_2)$ is finite;

- $\phi = \langle \mathcal{A} \rangle \phi_1$, we have that

$$\mathcal{R}_{\mathcal{N}}(\langle \mathcal{A} \rangle \phi) = \bigcup_{a \in \mathcal{A}(\mathcal{N})} \bigcup_{(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]} \mathcal{R}_{\mathcal{N}}(\phi\{\delta\}) \cup \{\langle \mathcal{A} \rangle \phi\}$$

  which is finite because:

  - $\mathcal{R}_{\mathcal{N}}(\phi_1\{\delta\}$ is finite from inductive hypothesis;
  - and $\mathcal{A}(\mathcal{N})$ is finite if $\mathcal{N}$ is finite.

- $\phi = \nu\kappa'.\phi_1$ in this cases we have that:

$$\mathcal{R}_{\mathcal{N}}(\nu\kappa'.\phi_1) = \mathcal{R}_{\mathcal{N}}(\phi_1)[\nu\kappa.\phi/\kappa'] \cup \{\nu\kappa.\phi_1\}$$

  which is finite from the inductive hypothesis.

- $\phi = \neg\neg\phi_1$, in this case $\mathcal{R}_{\mathcal{N}}(\neg\neg\phi_1) = \{\neg\neg\phi_1\} \cup \mathcal{R}_{\mathcal{N}}(\phi_1)$ which is finite for inductive hypothesis.

- the thesis follows analogously for the cases $\neg\langle \mathcal{A} \rangle \phi_1$, $\neg(\phi_1 \vee \phi_2)$ and $\neg\nu\kappa.\phi_1$.

□

Lemma 4.12 *Let ϕ be a closed formula, then for all ϕ' and δ:*

$$\phi'[\phi/\kappa]\{\delta\} = \phi'\{\delta\}[\phi/\kappa]$$

Proof: Lemma is proved by induction on the syntax of ϕ'.

Base of Induction If $\phi' = \kappa$, $\phi' = \kappa'$ with $\kappa \neq \kappa'$ or $\phi' = \mathtt{tp@}s$ then $\phi'[\phi/\kappa]\{\delta\} = \phi'\{\delta\}[\phi/\kappa]$

Inductive Hypothesis Let ϕ_1 and ϕ_2 be such that for every δ ($i \in \{1, 2\}$):

$$\phi_i[\phi/\kappa]\{\delta\} = \phi_i\{\delta\}[\phi/\kappa]$$

Inductive Step

- $\phi' = \phi_1 \wedge \phi_2$:

$$\begin{array}{rcl} \phi'[\phi/\kappa]\{\delta\} & = & (\phi_1 \wedge \phi_2)[\phi/\kappa]\{\delta\} \\ & = & \phi_1[\phi/\kappa]\{\delta\} \wedge \phi_1[\phi/\kappa]\{\delta\} \\ & = & \phi_1\{\delta\}[\phi/\kappa] \wedge \phi_1\{\delta\}[\phi/\kappa] \\ & = & (\phi_1 \wedge \phi_2)\{\delta\}[\phi/\kappa] \end{array}$$

- $\phi' = \langle\mathcal{A}\rangle\phi_1$:

$$\begin{aligned}\phi'[\phi/\kappa]\{\delta\} &= (\langle\mathcal{A}\rangle\phi_1)[\phi/\kappa]\{\delta\}\\ &= \langle\mathcal{A}\{\delta\}\rangle(\phi_1[\phi/\kappa]\{\delta'\})\\ &= \langle\mathcal{A}\{\delta\}\rangle(\phi_1\{\delta'\}[\phi/\kappa])\\ &= (\langle\mathcal{A}\{\delta\}\rangle\phi_1\{\delta'\})[\phi/\kappa]\\ &= (\langle\mathcal{A}\rangle\phi_1)\{\delta\}[\phi/\kappa]\end{aligned}$$

  where $\delta'(u) = \begin{cases} u & \text{if } ?u \text{ appears in } \mathcal{A} \\ \delta(u) & \text{otherwise} \end{cases}$.

- $\phi' = \neg\phi_1$:

$$\begin{aligned}\phi'[\phi/\kappa]\{\delta\} &= (\neg\phi_1)[\phi/\kappa]\{\delta\}\\ &= \neg(\phi_1[\phi/\kappa]\{\delta\})\\ &= \neg(\phi_1\{\delta\}[\phi/\kappa])\\ &= (\neg\phi_1)\{\delta\}[\phi/\kappa]\end{aligned}$$

- $\phi' = \nu\kappa'.\phi_1$: if $\kappa = \kappa'$ then $\phi'[\phi/\kappa]\{\delta\} = \phi'\{\delta\} = \phi'\{\delta\}[\phi/\kappa]$ else, if $\kappa \neq \kappa'$:

$$\begin{aligned}\phi'[\phi/\kappa]\{\delta\} &= (\nu\kappa'\phi_1)[\phi/\kappa]\{\delta\}\\ &= \nu\kappa'(\phi_1[\phi/\kappa]\{\delta\})\\ &= \nu\kappa'(\phi_1\{\delta\}[\phi/\kappa])\\ &= (\nu\kappa'\phi_1)\{\delta\}[\phi/\kappa]\end{aligned}$$

□

Lemma 4.13 *Let ϕ and ϕ' be formulae and let $\mathcal{N}$ be a set of nets, if ϕ' is closed then:*

$$\mathcal{R}_{\mathcal{N}}(\phi[\phi'/\kappa]) \subseteq \mathcal{R}_{\mathcal{N}}(\phi)[\phi'/\kappa] \cup \mathcal{R}_{\mathcal{N}}(\phi')$$

Proof: We prove the lemma by induction on the structure of ϕ.

Base of Induction. If $\phi = \texttt{tp@}s$, $\phi = \neg\texttt{tp@}s$ or $\phi = \kappa$ the thesis follows easily.

Inductive Hypothesis. We suppose that for ϕ_1 and ϕ_2 hold that (for $i \in \{1, 2\}$).

$$\mathcal{R}_{\mathcal{N}}(\phi_i[\phi'/\kappa]) \subseteq \mathcal{R}_{\mathcal{N}}(\phi_i)[\phi'/\kappa]) \cup \mathcal{R}_{\mathcal{N}}(\phi')$$

Inductive Step. We are going to prove for $\phi = \phi_1 \vee \phi_2$, $\phi = \langle\mathcal{A}\rangle\phi$ and $\phi = \nu\kappa.\phi'$. Cases $\phi = \neg(\phi_1 \vee \phi_2)$, $\phi = \neg\langle\mathcal{A}\rangle\phi$ and $\phi = \neg\nu\kappa.\phi'$ can be obtained analogously.

- $\phi = \phi_1 \vee \phi_2$, we have that

$$\begin{aligned}\mathcal{R}_{\mathcal{N}}((\phi_1 \vee \phi_2)[\phi'/\kappa]) &= \mathcal{R}_{\mathcal{N}}(\phi_1[\phi'/\kappa] \vee \phi_2[\phi'/\kappa])\\ &= \mathcal{R}_{\mathcal{N}}(\phi_1[\phi'/\kappa]) \cup \mathcal{R}_{\mathcal{N}}(\phi_2[\phi'/\kappa]) \cup \{\phi_1 \vee \phi_2\}\end{aligned}$$

  from the inductive hypothesis we also have that

$$\mathcal{R}_{\mathcal{N}}(\phi_i[\phi'/\kappa]) \subseteq \mathcal{R}_{\mathcal{N}}(\phi_i)[\phi'/\kappa]) \cup \mathcal{R}_{\mathcal{N}}(\phi')$$

then we have that

$$\mathcal{R}_{\mathcal{N}}(\phi_1[\phi'/\kappa]) \cup \mathcal{R}_{\mathcal{N}}(\phi_2[\phi'/\kappa]) \cup \{\phi_1 \vee \phi_2\} \subseteq$$
$$\{\phi_1[\phi'/\kappa] \vee \phi_2[\phi'/\kappa]\} \cup \mathcal{R}_{\mathcal{N}}(\phi_1)[\phi'/\kappa] \cup \mathcal{R}_{\mathcal{N}}(\phi_2)[\phi'/\kappa] \cup \mathcal{R}_{\mathcal{N}}(\phi')$$
$$= \mathcal{R}_{\mathcal{N}}(\phi_1 \vee \phi_2)[\phi'/\kappa] \cup \mathcal{R}_{\mathcal{N}}(\phi')$$

that is the thesis.

- $\phi = \langle \mathcal{A} \rangle \phi_1$, then

$$\mathcal{R}_{\mathcal{N}}(\langle \mathcal{A} \rangle \phi_1[\phi'/\kappa]) = \bigcup_{a \in \mathcal{A}(\mathcal{N})} \bigcup_{(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]} \mathcal{R}_{\mathcal{N}}(\phi_1[\phi'/\kappa]\{\delta\}) \cup \{\langle \mathcal{A} \rangle \phi[\phi'/\kappa]\} = (*)$$

For the previous lemma, we have that

$$\phi_1[\phi'/\kappa]\{\delta\} = \phi_1\{\delta\}[\phi'/\kappa]$$

then

$$\begin{array}{rcl}
(*) & = & \bigcup_{a \in \mathcal{A}(\mathcal{N})} \bigcup_{(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]} \mathcal{R}_{\mathcal{N}}(\phi_1\{\delta\}[\phi'/\kappa]) \cup \{\langle \mathcal{A} \rangle \phi[\phi'/\kappa]\} \\
& = & \bigcup_{a \in \mathcal{A}(\mathcal{N})} \left(\bigcup_{(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]} \mathcal{R}_{\mathcal{N}}(\phi_1\{\delta\})[\phi'/\kappa] \cup \mathcal{R}_{\mathcal{N}}(\phi') \right) \cup \{\langle \mathcal{A} \rangle \phi[\phi'/\kappa]\} \\
& = & \bigcup_{a \in \mathcal{A}(\mathcal{N})} \left(\bigcup_{(a;\delta) \in \mathbb{A}[\![\mathcal{A}]\!]} \mathcal{R}_{\mathcal{N}}(\phi_1\{\delta\})[\phi'/\kappa] \right) \cup \mathcal{R}_{\mathcal{N}}(\phi') \cup \{\langle \mathcal{A} \rangle \phi[\phi'/\kappa]\} \\
& = & \mathcal{R}_{\mathcal{N}}(\langle \mathcal{A} \rangle \phi)[\phi'/\kappa] \cup \mathcal{R}_{\mathcal{N}}(\phi')
\end{array}$$

- $\phi = \nu\kappa'.\phi_1$, we have that:

$$\mathcal{R}_{\mathcal{N}}(\nu\kappa'.\phi_1[\phi'/\kappa]) = \mathcal{R}_{\mathcal{N}}(\phi_1[\phi'/\kappa])[\phi_1/\kappa'] \cup \{\nu\kappa.\phi_1[\phi'/\kappa]\}$$

from the inductive hypothesis follows that

$$\mathcal{R}_{\mathcal{N}}(\phi_1[\phi'/\kappa])[\phi_1/\kappa'] = \mathcal{R}_{\mathcal{N}}(\phi_1)[\phi'/\kappa][\phi_1/\kappa'] \cup \mathcal{R}_{\mathcal{N}}(\phi')[\phi_1/\kappa]$$

for hypothesis ϕ' is closed then

$$\mathcal{R}_{\mathcal{N}}(\phi_1[\phi'/\kappa])[\phi_1/\kappa'] = \mathcal{R}_{\mathcal{N}}(\phi_1)[\phi_1/\kappa'][\phi'/\kappa] \cup \mathcal{R}_{\mathcal{N}}(\phi')$$

□

Corollary 4.14 *Let $\nu\kappa.\phi$ be a formula and $\mathcal{N}$ a set of nets then $\mathcal{R}_{\mathcal{N}}(\phi[\nu\kappa.\phi/\kappa]) = \mathcal{R}_{\mathcal{N}}(\phi)[\nu\kappa.\phi/\kappa] \cup \{\nu\kappa.\phi\}$*

Corollary 4.15 *Let N be a net and let $\mathcal{N} = \{N' | N \succ\!\!\rightarrow^* N'\}$. Let $\pi_1 = H_1 \vdash N_1 : \phi_1$ and $\pi_2 = H_2 \vdash N_2 : \phi_2$ such that: $N_1, N_2 \in \mathcal{N}$ and $\pi_1 \sqsubseteq_I \pi_2$, then:*

$$\mathcal{R}_{\mathcal{N}}(\phi_2) \subseteq \mathcal{R}_{\mathcal{N}}(\phi_1)$$

Moreover, for every formula ϕ and every set of hypothesis H:

$$\{\phi' | \exists H', N'.H \vdash N : \phi \sqsubseteq H' \vdash N' : \phi'\} \subseteq \mathcal{R}_{\mathcal{N}}(\phi)$$

Lemma 4.16 *Let N be a net such that the set $\mathcal{N} = \{N' | N \succ\!\!\rightarrow^* N'\}/\equiv$ is finite, ϕ be a closed formula and H be a set of hypothesis, then every sequence $\{\pi_i\}_{i\in I}$ starting from $\pi = H \vdash N : \phi$ is finite.*

Proof: We have that the set of formulae that appear in the sequence is finite. Indeed, it is a subset of $\mathcal{R}_{\mathcal{N}}(\phi)$ which is finite for Lemma 4.11. Let

$$\Phi = \{\phi' | \exists i.\pi_i = H' \vdash N' : \phi'\}$$

We prove theorem by induction on the size n of Φ.

Base of Induction. Let $n = 1$, and let ϕ then only one formula that appears in Π. We can distinguish two cases:

- $\phi = \texttt{tp@}s$, $\phi = \neg\texttt{tp@}s$, $\phi = tt$ or $\phi = \nu\kappa.\phi$ or $\phi = \neg\nu\kappa.\phi$ and $N : \nu\kappa.\phi \in H$. In this case there are not a sequent π' such that $\pi \sqsubseteq_I \pi'$. Every sequence starting from π is obviously finite.
- $\phi = \nu\kappa.\kappa$, then for every N and H the only (finite) sequence from $H \vdash N : \phi$ is the following:

$$H \vdash N : \nu\kappa.\kappa \sqsubseteq_I H \cup \{N : \nu\kappa.\kappa\} \vdash N : \nu\kappa.\kappa$$

Inductive Hypothesis. We suppose that if size of Φ is less or equal to n then π has only finite sequences.

Inductive Step. Let $\pi = H \vdash N : \phi$ such that size of Φ is $n + 1$. We can distinguish two cases.

- There isn't in Φ a formula like $\nu\kappa.\phi$. In this case we have that, for every i, ϕ_{i+i} is a strict subterm of ϕ_i (up to a substitution). This means that every formulae occur exactly one time in the sequence. For inductive hypothesis we have that the sequence that starts from π_1 is finite. Then also the sequence that starts from π_0 is finite.
- There exists $\nu\kappa.\phi'$ in Φ. Let $\nu\kappa.\phi_1$ such that it does not exist $\nu\kappa'.\phi' \in \Phi$ such that $\nu\kappa'.\phi' \prec \nu\kappa.\phi_1$. Let π_i be first sequent in which compare $\nu\kappa.\phi_1$. For every $\pi_j = H_j \vdash N_j : \phi_j$ and $\pi_i \sqsubseteq \pi_j\ N_i : \nu\kappa.\phi_1 \in H_j$ than the size of $\{\pi_j = H_j \vdash N_j : \phi_j | j \in I \& \pi_i \sqsubseteq \pi_j\}$ is less or equal to the size of $\mathcal{N}$, let $\pi_{\hat{j}} = H_{\hat{j}} \vdash N_{\hat{j}} : \nu\kappa.\phi_1$ be its greatest element. If $\pi_{\hat{j}}$ is a successful sequents, then the sequence is finite; otherwise there exists π' such that $\pi_{\hat{j}} \sqsubseteq_I \pi'$ and

  $$|\{\phi' | \exists i.\pi' \sqsubseteq_I \pi_i = H' \vdash N' : \phi'\}| \leq n$$

  then, from inductive hypothesis, every sequence from π' is finite then also $\{\pi_i\}_{i\in I}$ is finite.

□

Theorem 4.17 *For every net N such that the set $\mathcal{N} = \{N' | N \succ\!\!\rightarrow^* N'\}/\equiv$ is finite, for every closed formula ϕ and for every set of hypothesis H, the prove for $H \vdash N : \phi$ is finite.*

Proof: We have that $\{\pi | H \vdash N : \phi \sqsubseteq_I \pi\}$ is finite (indeed $\mathcal{N}$ is finite), then if Π is an infinite proof for $H \vdash N : \phi$ then there exists an infinite sequence $\{\pi_i\}_{i\in I}$ from $H \vdash N : \phi$ which is in contradiction with Lemma 4.16 □

Definition 4.18 *Let N be such that the set $\mathcal{N} = \{N' | N \succ\!\!\rightarrow^* N'\}/equiv$ is finite, and let a formula ϕ and a set of hypothesis H. We define* $\min(H \vdash N : \phi)$ *as the least length of a prove Π for the sequent $H \vdash N : \phi$.*

4.3.3 Completeness

In order to prove the completeness, first we prove that, if the set of reachable nets from N is finite either $H \vdash N : \phi$ or $H \vdash N : \neg\phi$ is provable.

Lemma 4.19 *Let N be such that has a finite set of reachable nets, then either the sequent $H \vdash N : \phi$ or the sequent $H \vdash N : \neg\phi$ are provable.*

Proof: The lemma is provable by induction on $\min(H \vdash N : \phi)$ as the Lemma 3.14. □

Please, notice that if the set of nets reachable form N is not finite then the previous lemma does not hold. Indeed, we are not able to prove properties like $\nu\kappa.\phi \wedge [\circ]\kappa$ for which we need to *explore* all the reachable nets.

Theorem 4.20 *Let N be such that the set of reachable nets is finite and let ϕ be a formula. If $N \in \mathbb{M}[\![\phi]\!]^H \epsilon_0 \delta_0$ then $H \vdash N : \phi\{\delta\}$ is provable.*

Proof: We suppose that $N \in \mathbb{M}[\![\phi]\!]^H \epsilon\delta$ and that $H \vdash N : \phi\{\delta\}$ is not provable. For the previous lemma we have that $H \vdash N : \neg\phi\{\delta\}$ is provable, and from the soundness we have also that $N \in \mathbb{M}[\![\neg\phi]\!]^H \epsilon\delta$. Thus from the definition of $\mathbb{M}[\![\cdot]\!]^H$ we have that $N \in Net - \mathbb{M}[\![\phi]\!]^H \epsilon\delta$ which is in contradiction with the hypothesis. □

4.3.4 Completeness for infinite systems

In the previous section we showed that completeness for the proof system can be guaranteed only for a subset of the KLAIM nets.

In this section, we present a subset of formulae for which we are able to prove a completeness result. These formulae, that we call *regular*, are those that permit specifying *eventually-properties*.

In order to simplify definition of such formulae we will explicitly use in the syntax formulae like $\phi_1 \wedge \phi_2$, $[\mathcal{A}]\phi$ and $\mu\kappa.\phi$. In previous sections we shown how to express these formulae by using our original language and can be thought as macro. Consequently we have to modify the rules $R1$, $R5$ and $R6$ in Table 15 as follows:

$$R1 \qquad \frac{H \vdash N : \phi_1 \qquad H \vdash N : \phi_2}{H \vdash N : \phi_1 \wedge \phi_2}$$

$$R5 \qquad \frac{H \vdash N_1 : \phi\{\delta_1\} \quad H \vdash N_2 : \phi\{\delta_2\} \quad \dots}{H \vdash N : [\mathcal{A}]\phi} \left[\forall (a_i; \delta_i) \in \mathbb{A}[\![\mathcal{A}]\!] : \forall N_i \in \left\{N' | N \succ\!\!\xrightarrow{a_i} N'\right\}\right]$$

$$R6 \qquad \frac{H' \cup \{N : \mu\kappa.\phi\} \vdash N : \phi[\mu\kappa.\phi/\kappa]}{H \vdash N : \mu\kappa.\phi} \; [N' \equiv N, N' : \mu\kappa.\phi \notin H]$$

Definition 4.21 *We say that a formula ϕ is* regular *if it is generated by:*

$$\phi ::= \mathbf{tt} \mid \mathbf{ff} \mid tp@\sigma \mid \neg tp@\sigma \mid \langle\mathcal{A}\rangle\phi \mid [\mathcal{A}]\phi \mid \phi \vee \phi \mid \phi_1 \wedge \phi_2 \mid \kappa \mid \mu\kappa.\phi$$

Lemma 4.22 *If ϕ is regular and it does not have a subformula $\mu\kappa.\phi'$ then for every net N if $N \in [\![\phi]\!]^H$ then $H \vdash N : \phi$ is provable.*

Proof: The lemma follows directly from completeness of formulae without recursion. □

Definition 4.23 *Let $\mu\kappa.\phi$ a closed formula. We define $\widetilde{\phi}_i$, as follows:*

$$\widetilde{\phi}_0 = \mathbf{ff}$$

$$\widetilde{\phi}_{i+1} = \phi[\widetilde{\phi}_i/\kappa]$$

Lemma 4.24 *Let $\mu\kappa.\phi$ be a closed formula and let N be a net:*

$$N \in \mathbb{M}[\![\mu\kappa.\phi]\!] \Leftrightarrow \exists i.\ N \models \widetilde{\phi}_i$$

Lemma 4.25 *For every formulae ϕ' ($\phi' \neq \kappa$), and $\mu\kappa.\phi$: for every i if $H \vdash N : \phi'[\widetilde{\phi}_i/\kappa]$ is provable then $H \vdash N : \phi'[\mu\kappa.\phi/\kappa]$ is provable too.*

Proof: We prove the lemma by induction on the length n of the proof for $H \vdash N : \phi'[\widetilde{\phi}_i/\kappa]$.

Base of Induction $n = 0$ in this case $N : \phi'[\widetilde{\phi}_i/\kappa]$ is a successful sequent. Then $\phi'[\widetilde{\phi}/\kappa] = \mathtt{tp@}s$, $\phi'[\widetilde{\phi}/\kappa] = \neg\mathtt{tp@}s$. In all cases also $H \vdash N : \phi'[\mu\kappa.\phi/\kappa]$ is a successful sequent.

Base of Induction We suppose that, for every formula ϕ', if $H \vdash N : \phi'[\widetilde{\phi}_i/\kappa]$ is provable in at most n steps then $H \vdash N : \phi'[\mu\kappa.\phi/\kappa]$ is provable too.

Inductive Step Let $H \vdash N : \phi'[\widetilde{\phi}_i/\kappa]$ be provable in $n+1$ steps. We distinguish according to first applied rule.

- $R1$ In this case we have that:

$$\frac{H \vdash N : \phi_1[\widetilde{\phi}/\kappa] \qquad H \vdash N : \phi_2[\widetilde{\phi}/\kappa]}{H \vdash N : \phi_1[\widetilde{\phi}_i/\kappa] \wedge \phi_2[\widetilde{\phi}_i/\kappa])}$$

  We have that if $\phi_i \neq \kappa$ then the thesis follows for inductive hypothesis, otherwise $\phi_1[\widetilde{\phi}_i/\kappa] = \phi[\phi_{i-1}/\kappa]$ (i has to be greatest then 0) which is provable with at most n steps. By applying inductive hypothesis we have that $(\phi_1 \wedge \phi_2)[\mu\kappa.\phi/\kappa]$ is provable too.

- for $R2$, $R3$ and $R5$[1] we can proceed as for the previous case.

- $R6$ In this case we have that:

$$\frac{H' \cup \{N : \mu\kappa'\phi'[\widetilde{\phi}_i/\kappa]\} \vdash N : \phi'[\mu\kappa'.\phi'/\kappa'][\widetilde{\phi}/\kappa]}{H \vdash N : \mu\kappa'.\phi'[\widetilde{\phi}_i/\kappa]}$$

  by using inductive hypothesis we have that

$$H' \cup \{N : \mu\kappa'\phi'[\widetilde{\phi}_i/\kappa]\} \vdash N : \phi'[\mu\kappa'.\phi'/\kappa'][\mu\kappa.\phi/\kappa]$$

  is provable.

[1] If ϕ is regular rule $R4$ never will applied in any proof for ϕ

□

Theorem 4.26 *If ϕ is regular then, for every net N if $N \in [\![\phi]\!]^H$ then $H \vdash N : \phi$ is provable.*

Proof: The proof follows by using previous two lemmata. □

5 Using the logic

In this section, we will present three examples that should help convincing the reader of the qualities of the proposed logic. The first example, which has been presented in Section 2.2, shows how to prove properties for a simple KLAIM net. The second one shows how the logic can be used for specifying access policies of KLAIM nets. The final one formalizes properties for a more complex system managing distribute information.

5.1 The itinerant agent

In this section we present a simple proof built by using our proof system. Let N be the net defined as follows:

$$s_1 :: Proc_1 | \mathbf{out}(s_2) \parallel s_2 :: \mathbf{nil}$$

where

$$Proc_1 = \mathbf{in}(!u)@\mathtt{self}.\mathbf{eval}(Proc_1)@u.\mathbf{out}(\mathtt{self})@u.\mathbf{nil}$$

In Section 2.2 we have presented the computation of this net.

For this net we can prove, by using the proof systems, that:

never tuple (s_1) will be at site s_2 and, at the same time, (s_2) will be at site s_1

This property can be formally stated as follows:

$$\phi = \nu\kappa.\,((\neg(s_1)@s_2) \vee (\neg(s_2)@s_1)) \wedge [\circ]\kappa$$

The proof for $\vdash N : \phi$ starts with the application of rule $R7$, then rule $R1$ is applied:

$$\dfrac{\dfrac{N : \phi \vdash N : \neg(s_1)@s_2) \vee (\neg(s_2)@s_1) \qquad N : \phi \vdash N : [\circ]\phi}{N : \phi \vdash N : ((\neg(s_1)@s_2) \vee (\neg(s_2)@s_1)) \wedge [\circ]\phi}\, R1}{\vdash N : \phi}\, R7$$

We can prove $N : \phi \vdash N : \neg((s_1)@s_2) \vee \neg((s_2)@s_1)$ by applying rule $R2$ and obtaining the successful sequent $N : \phi \vdash N : \neg((s_1)@s_2)$. Please notice that for each net N_i reachable from N (see Section 2.2) and for all H the sequent $H \vdash N_i : \neg((s_1)@s_2) \vee \neg((s_2)@s_1)$ is provable.

The other branch of the proof, $N : \phi \vdash N : [\circ]\phi$, proceeds with the application of rule $R5$. After this rules $R7$ and $R1$ are applied again to obtain:

$$\dfrac{\dfrac{\dfrac{N_1 : \phi, N : \phi \vdash N_1 : \neg(s_1)@s_2) \vee (\neg(s_2)@s_1) \qquad N_1 : \phi, N : \phi \vdash N_1 : [\circ]\phi}{N_1 : \phi, N : \phi \vdash N_1 : ((\neg(s_1)@s_2) \vee (\neg(s_2)@s_1)) \wedge [\circ]\phi}\, R1}{N : \phi \vdash N_1 : \phi}\, R7}{N : \phi \vdash N : [\circ]\phi}\, R5$$

Like in previous case, we proceed by applying rule $R2$ for one branch and rule $R7$ for the other. By proceeding analogously on the obtained sequent we obtain the successful sequent:

$$N : \phi, N_1 : \phi, N_2 : \phi, N_3 : \phi, N_4 : \phi, N_5 : \phi, N_6 : \phi \vdash N : \phi$$

5.2 Access right specifications

In this section, we show how to use our logic for specifying and verifying access rights in a KLAIM net. To specify that *"no process is ever evaluated at a site s"* we can use the following formulae:

$$\nu\kappa.[\mathtt{E}(?u, \mathtt{1_P}, s)]\mathbf{ff} \wedge [\circ]\kappa$$

We can also specify dynamic access policies. Let N be a net with two sites s_1 and s_2. In that net we want guarantee that *"s_1 doesn't insert tuples at site s_2 after s_3 has evaluated a process at s_1"*. We can formalize this properties as follows:

$$\nu\kappa.[\mathtt{E}(s_3, \mathtt{1_P}, s_1)]\left(\nu\kappa'.[\mathtt{O}(s_1, \mathtt{1_t}, s_2)]\mathbf{ff} \wedge [\circ]\kappa'\right) \wedge [\circ]\kappa$$

Composing formulae we obtain more complex access specifications. Let N' be another net with two sites s_1 and s_2. We establish that *"s_1 cannot read tuples containing a pair* (locality,process) *at s_2, and s_2 cannot evaluate any process at s_1 that reads tuples that contain a locality and puts these values at s_1"*. To specify these access rights we can use the following formula:

$$\nu\kappa.[\mathtt{R}(s_1, ?u, \mathtt{1_P}), s_2)]\mathbf{ff} \wedge [\mathtt{R}(s_1, ?l, \mathtt{1_P}), s_2)]\mathbf{ff} \wedge$$
$$[\mathtt{E}(s_2, \mathtt{r}(!u)@s_1 \rightarrow \mathtt{o}(u), s_1) \cup \mathtt{E}(s_2, \mathtt{i}(!u)@s_1 \rightarrow \mathtt{o}(u), s_1)]\mathbf{ff} \wedge [\circ]\kappa$$

We can also specify access rights that depend on properties of nodes. Suppose that in a net we require that each node s_1 can evaluate a processes at a node s_2 only if tuple (s_1) is in the tuple space at s_2. This properties can be rendered as follows:

$$\nu\kappa.[\mathtt{E}(?u_1, \mathtt{1_P}, ?u_2)](u_1)@u_2 \wedge [\circ]\kappa$$

5.3 Distributed Information Systems management

In this section, we consider a larger example of a Distributed Information Systems management. We assume that a database is distributed over three different sites, named, Inf_i $i \in \{1, 2, 3\}$. A node, named *Manager*, manages the database system sending processes to sites for updating the information. Only one updating-process at a time can be executed at a site. For this reason, inside the tuple space of Inf_i there is the tuple $''F''$. An updating process can be evaluated at Inf_i only if tuple $''F''$ is in its tuple space.

The net of the distributed database is defined as follows:

$$Inf_1 ::_\rho \mathbf{out}(''F'') \parallel Inf_2 ::_\rho \mathbf{out}(''F'') \parallel Inf_3 ::_\rho \mathbf{out}(''F'')$$

In the tuple space of node *Manager* there is a tuple $(''G'')$ for each node Inf_i. An updating process can be started only when at least a tuple $(''G'')$ is in the tuple space of *Manager*.

Process *StartAgent* looks for a tuple $(''G'')$. When this is found, *StartAgent* invokes *CallUpdate*, which starts the updating procedure. By guarding *CallUpdate* in *StartAgent* with an $\mathbf{in}(''G'')@\mathtt{self}$ action we guarantee that the system is deadlock free.

$$\begin{aligned} StartAgent &= \mathbf{in}(''G'')@\mathtt{self}.\quad (CallUpdate(Inf_1, Inf_2, Inf_3) \\ &\qquad\qquad |StartAgent) \end{aligned}$$

$$\begin{aligned} CallUpdate\langle u_1, u_2, u_3\rangle &= \mathbf{in}(''F'')@u_1.\mathbf{out}(''updating'')@u_1. \\ &\qquad \mathbf{eval}(Update(u_2, Update(u_3, FUpdate(Manager))))@u_1.\mathbf{nil} \end{aligned}$$

$$\begin{aligned} Update\langle u, X\rangle &= \mathbf{in}(''F'')@u.\mathbf{out}(''updating'')@u.\mathbf{eval}(X)@u. \\ &\qquad \mathbf{in}(''updating'')@\mathtt{self}.\mathbf{out}(''F'')@\mathtt{self}.\mathbf{nil} \end{aligned}$$

$$FUpdate\langle u\rangle = \mathbf{in}(''updating'')@\mathtt{self}.\mathbf{out}(''F'')@\mathtt{self}.\mathbf{eval}(Success)@u.\mathbf{nil}$$

$$Success = \mathbf{out}(''G'')@\mathtt{self}.\mathbf{nil}$$

The manager node is define as follows:

$$\begin{aligned} Manager :: \; & StartAgent|\mathbf{out}(Inf_1)|\mathbf{out}(Inf_2)|\mathbf{out}(Inf_3) \\ & |\mathbf{out}(''G'')|\mathbf{out}(''G'')|\mathbf{out}(''G'') \end{aligned}$$

We would like to prove that, within the above system, if a process is evaluated at site Inf_i then no process is evaluated at the same site until the updating terminates. This property is specified by the formula ϕ_1 below:

$$\phi_1 = \nu\kappa.[\mathbf{e}(?u_1, \mathtt{1_P}, ?u_2) - \mathbf{e}(?u_3, \mathtt{1_P}, Manager)]\phi_2 \wedge [\circ]\kappa$$

where

$$\phi_2 = \nu\kappa'.[\mathbf{e}(?u, \mathtt{1_P}, u_2)]\mathbf{ff} \wedge [\mathbf{e}(u_2, \mathtt{1_P}, ?u_4)]\mathbf{tt} \vee [\circ]\kappa'$$

The property can be proved by using our proof system. However, the proof is, in same sense, *automatic* and requires large space to be presented. For this reasons we omit it.

6 Conclusions and Related Works

In this paper we have presented a variant of HML enriched with more refined action predicates and state formulae. The logic is tailored for reasoning about properties of KLAIM systems. The proposed logic has been equipped with a proof system based on tableau. This system has been inspired by [6, 18, 19].

The main differences of our solution with respect to the existing ones, also based on Hennessy-Milner logic, reside on the different use of transition labels. In the *standard* approaches, even those considering value passing [16], labels are considered as *basic entities* and are characterized, inside modal operators, syntactically. In our approach transition labels are characterized by means of their *properties*.

The definition of the proof system is, in some sense, standard. However, since we have explicit notion of values, proofs of *completeness* and *soundness* results are more complex.

There are a few papers that have tackled the problem of defining a logic for process calculi with primitives for mobility. More specifically they have considered definitions of logics for $\pi-$calculus [15] and Mobile Ambients [2].

In [14] a modal logic for π-calculus has been introduced. This logic aims at capturing different equivalences between processes and at establishing the differences between late and early bisimulation. Also this logic is based on Hennessy-Milner Logic, but it has no operators for recursion. In [7] a HML-like logic with recursion for π-calculus is presented. This logic is equipped with a tableau-based proof system. However, neither [14] nor [7] consider mobility/spatial features.

In [3] a modal logic for Mobile Ambients [2] has been presented; in [1] a variant of the Ambient logic, tailored for asynchronous π-calculus, is introduced. This very interesting logic is equipped with operators for spatial and temporal properties, and for *compositional* specification of systems properties. In [4], new operators for expressing logical properties of name restrictions have been added. This logic permits describing properties of spatial configurations and of mobile computation, including security properties.

All of these properties are essentially *state based*, in the sense that systems are proved to be safe/unsafe by only taking into account spatial configurations. The structure of the system can be formalized in details: for instance one can *count* the resources in the system. However, any information about the history of an ambient in the system is lost. For instance properties like: *"an ambient can enter into n only after another one exits from n"* cannot be expressed. In sections 5 we showed how similar properties for KLAIM systems can be expressed in our logic.

Sangiorgi, in [17], shows how to express, in the ambient logic properties like *"an ambient n is opened inside the ambient m"*. However, it is not possible to univocally identify an ambient in the system. A precise advantage of our logic is that it allows us to talk about specific, unique sites, whereas, since the ambient calculus allows multiple ambients to have the same name, it is difficult to talk about a specific ambient in the ambient logic. Moreover, to express *evolutionary properties* Sangiorgi needs to use logical operators that, in general, are not decidable. Indeed, the Ambient logic is not completely decidable and a sound and complete proof system is available only for a subset of the logic.

Acknowledgments We are grateful to the anonymous referees, whose useful comments helped us to improve the presentation. Thanks to Lorenzo Bettini and Betti Venneri for their helpful comments.

References

[1] L. Caires and L. Cardelli. A spatial logic for concurrency (Part I). In N. Kobayashi and B. C. Pierce, editors, *Proceeding of Theoretical Aspects of Computer Software; 4th International Symposium, TACS 2001*, number 2215 in Lecture Notes in Computer Science, pages 1–37, October 2001.

[2] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of the First International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. sv, 1998.

[3] L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *27th Annual Symposium on Principles of Programming Languages (POPL) (Boston, MA)*. ACM, 2000.

[4] L. Cardelli and A. D. Gordon. Logical properties of name restriction. In *Proceeding of International Conference on Typed Lambda Calculi and Applications, TLCA'01*, volume 2044 of *Lecture Notes in Computer Science*. Springer, 2001.

[5] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(10):444–458, October 1989. Technical Correspondence.

[6] R. Cleaveland. Tableau-based model checking in the propositional μ-calculus. *Acta Informatica*, 27(8):725–747, September 1990.

[7] M. Dam. Model checking mobile processes. *Journal of Information and Computation*, 129(1):35–51, 1996.

[8] R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for Access Control. *Theoretical Computer Science*, 240(1):215–254, 2000.

[9] R. De Nicola, G. L. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998.

[10] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[11] D. Gelernter. Multiple Tuple Spaces in Linda. In J. G. Goos, editor, *Proceedings, PARLE '89*, volume 365 of *Lecture Notes in Computer Science*, pages 20–27, 1989.

[12] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, Jan. 1985.

[13] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.

[14] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[15] R. Milner, J. Parrow, and J. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40, 41–77, 1992.

[16] J. Rathke and M. Hennessy. Local model checking for value-passing processes. In M. Abadi and T. Ito, editors, *Proceeding of International Symposium on Theoretical Aspects of Computer Software, TACS'97*, Lecture Notes in Computer Science, pages 250 – 265. Springer-Verlag, 1997.

[17] D. Sangiorgi. Extensionality and intensionality of the ambient logics. In *28th Annual Symposium on Principles of Programming Languages (POPL) (London, UK)*, pages 4–13. ACM, Jan. 2001.

[18] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, Oct. 1991.

[19] G. Winskel. A note on model checking the modal ν-calculus. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 761–772, Berlin, July 1989. Springer.