Logical Methods in Computer Science
Vol. 4 (3:4) 2008, pp. 1–44
www.lmcs-online.org

Submitted Nov. 8, 2007
Published Sep. 4, 2008

SEPARABILITY IN THE AMBIENT LOGIC *

DANIEL HIRSCHKOFF [a], ÉTIENNE LOZES [b], AND DAVIDE SANGIORGI [c]

[a] ENS Lyon, Université de Lyon, CNRS, INRIA – France
e-mail address: Daniel.Hirschkoff@ens-lyon.fr

[b] LSV, ENS Cachan, CNRS – France
e-mail address: lozes@lsv.ens-cachan.fr

[c] Università di Bologna – Italy
e-mail address: Davide.Sangiorgi@cs.unibo.it

Abstract. The *Ambient Logic* (AL) has been proposed for expressing properties of process mobility in the calculus of Mobile Ambients (MA), and as a basis for query languages on semistructured data.

We study some basic questions concerning the discriminating power of AL, focusing on the equivalence on processes induced by the logic ($=_L$). As underlying calculi besides MA we consider a subcalculus in which an image-finiteness condition holds and that we prove to be Turing complete. Synchronous variants of these calculi are studied as well.

In these calculi, we provide two operational characterisations of $=_L$: a coinductive one (as a form of bisimilarity) and an inductive one (based on structual properties of processes). After showing $=_L$ to be stricly finer than barbed congruence, we establish axiomatisations of $=_L$ on the subcalculus of MA (both the asynchronous and the synchronous version), enabling us to relate $=_L$ to structural congruence. We also present some (un)decidability results that are related to the above separation properties for AL: the undecidability of $=_L$ on MA and its decidability on the subcalculus.

1. Introduction

This paper is devoted to the study of the *Ambient Logic* [14] (AL), a modal logic for expressing properties of Mobile Ambients [13] (MA) processes. The model of Mobile Ambients is based on the notion of locality (an ambient is a named locality), and interaction in MA appears as movement of localities. Localities may be nested, as in $a[P \mid b[Q] \mid c[R]]$, which describes an ambient a containing a process P as well as two sublocalities named b and c.

1998 ACM Subject Classification: F.3.2, F.4.1.

Key words and phrases: Process algebra, modal logic, Mobile Ambients, spatial logic.

* This work is a revised and extended version of parts of [30] and [20] (precisely, those parts that deal with issues related to separability).

[a] Work supported by the french projects ACI GEOCAL and ANR CHoCo.

[c] Work supported by european project Sensoria, italian MIUR Project n. 2005015785, "Logical Foundations of Distributed Systems and Mobile Code".

DOI:10.2168/LMCS-4 (3:4) 2008

An ambient can be thought of as a labelled tree. The sibling relation on subtrees represents spatial contiguity; the subtree relation represents spatial nesting. A label may represent an ambient name or a capability; moreover, a replication tag on labels indicates the resources that are persistent. The trees are unordered: the order of the children of a node is not important. As an example, the process $P \stackrel{\text{def}}{=} !a[\,\mathsf{in}\ c\,] \mid \mathsf{open}\ a.\,b[\,\mathbf{0}\,]$ can be thought of as a tree with $\mathsf{open}\ a.\,b[\,\mathbf{0}\,]$ on the roots node and $\mathsf{in}\ c$ on a child node labeled with a. The replication $!a$ indicates that the resource $a[\,\mathsf{in}\ c\,]$ is persistent: unboundedly many such ambients can be spawned. By contrast, $\mathsf{open}\ a$ is ephemeral: it can open only one ambient.

Syntactically, each tree is finite. Semantically, however, due to replications, a tree is an infinite object. As a consequence, the temporal developments of a tree can be quite rich. The process P above (we freely switch between processes and their tree representation) has only one reduction, to $\mathsf{in}\ c \mid !a[\,\mathsf{in}\ c\,] \mid b[\,\mathbf{0}\,]$. However, the process $!a[\,\mathsf{in}\ c\,] \mid !\mathsf{open}\ a.\,b[\,\mathbf{0}\,]$ can evolve into any process of the form

$$\mathsf{in}\ c \mid \ldots \mid \mathsf{in}\ c \mid b[\,\mathbf{0}\,] \mid \ldots \mid b[\,\mathbf{0}\,] \mid !a[\,\mathsf{in}\ c\,] \mid !\mathsf{open}\ a.\,b[\,\mathbf{0}\,]\,.$$

In general, a tree may have an infinite temporal branching, that is, it can evolve into an infinite number of trees, possibly quite different from each other (for instance, pairwise behaviourally unrelated). Technically, this means that the trees are not *image-finite*, where image-finite indicates a finiteness on the temporal branching of a process (we will come back to the definition of image-finiteness later).

Although the MA calculus often includes name restriction, $(\nu n)P$, reminiscent of the pi-calculus, we will omit this construction (unless we mention it explicitly), and will refer to public MA, or simply MA, for the calculus without name restriction.

In summary, MA is a calculus of dynamically-evolving unordered edge-labelled trees. AL is a logic for reasoning on such trees. The actual definition of satisfaction of the formulas is given on MA processes quotiented by a relation of *structural congruence*, $\equiv$, which equates processes with the same tree representation. (This relation is similar to Milner's structural congruence for the π-calculus [28].)

AL has also been advocated as a foundation of query languages for semistructured data [9]. Here, the laws of the logic are used to describe query rewriting rules and query optimisations. This line of work exploits the similarities between dynamically-evolving edge-labelled trees, underlying the ambient computational model, and standard models of semistructured data.

AL has a connective that talks about *time*, that is, how processes can evolve. The formula $\Diamond\,\mathcal{A}$ is satisfied by those processes with a future in which $\mathcal{A}$ holds. The logic has also connectives that talk about *space*, that is, the shape of the edge-labelled trees that describe process distributions. the formula $n[\,\mathcal{A}\,]$ is satisfied by ambients named n whose content satisfies $\mathcal{A}$ (read on trees: $n[\,\mathcal{A}\,]$ is satisfied by the trees whose root has just a single edge n leading to a subtree that satisfies $\mathcal{A}$); the formula $\mathcal{A}_1 \mid \mathcal{A}_2$ is satisfied by the processes that can be decomposed into parallel components P_1 and P_2 where each P_i satisfies $\mathcal{A}_i$ (read on trees: $\mathcal{A}_1 \mid \mathcal{A}_2$ is satisfied by the trees that are the juxtaposition of two trees that respectively satisfy the formulas $\mathcal{A}_1$ and $\mathcal{A}_2$); the formula 0 is satisfied by the terminated process $\mathbf{0}$ (on trees: 0 is satisfied by the tree consisting of just the root node).

AL is quite different from standard modal logics. First, the latter logics do not talk about space. Secondly, they have more precise temporal connectives. The only temporal connective of AL talks about the many-step evolution of a system on its own. In standard modal logics, by contrast, the temporal connectives also talk about the potential interactions between a process and its environment. For instance, in the Hennessy-Milner logic [18], the temporal modality $\langle\mu\rangle.\mathcal{A}$ is satisfied by the processes that can perform the action μ and become a process that satisfies $\mathcal{A}$. The action μ can be a reduction, but also an input or an output.

In this paper we study the equivalence between MA processes induced by the logic, written $=_L$: we write $P{=_L}Q$ if P and Q satisfy exactly the same formulas. Our main goal is to understand how much the logic discriminates between processes, i.e., to study the separating power of $=_L$. We show that $=_L$ is a rather fine-grained relation. Related to the problem of the equivalence induced by the logic are issues of decidability, that we also investigate.

The central technical device we rely on to analyse $=_L$ is a characterisation as a form of bisimilarity, that we call *intensional bisimilarity* and write $\simeq_{\text{int}}$. The bisimulation game defining $\simeq_{\text{int}}$ takes into account the interaction possibilities of agents, and also includes clauses to observe the spatial structure of processes, corresponding to the logical connectives of emptyness, spatial conjunction, and ambient. Intensional bisimilarity is to AL what standard bisimilarity is to Hennessy-Milner logic. In particular, $\simeq_{\text{int}}$ can be used to assess separability and expressiveness properties of the modal logic it captures. For instance, the definition of $\simeq_{\text{int}}$ reveals that, in some cases, logical observations are unable to distinguish between an agent entering an ambient, and the same agent going in and out of this ambient before finally entering it. We call this phenomenon *stuttering*. Stuttering can be seen as the spatial counterpart of the following 'eta law' for the asynchronous π-calculus [31]:

$$a(x).\,\big(\,\overline{a}\langle x\rangle \;\mid\; a(x).\,P\,\big) \;=\; a(x).\,P$$

(a similar equality also holds for communication in MA). Indeed, stuttering disappears when the asynchronous movements are replaced by synchronous ones, as is the case, e.g., in the model of Safe Ambients [25].

Something worth stressing is that our characterisation results are established on the full, public, MA calculus in which, as mentioned earlier, terms need not be image-finite, and with respect to a finitary logic. We are not aware of other results of this kind: characterisation results for a bisimilarity with respect to a modal logic in the literature (precisely, the completeness part of the characterisations) rely either on an image-finiteness hypothesis for the terms of the language, or on the presence of some infinitary constructs (such as infinitary conjunctions) in the syntax of the logic. Technically, the proof of our result is based on the definition of some complex modal formulas. To make it easier to understand our approach, we first present the main structure of the proof in a subcalculus without infinite behaviours; we then move to the full public MA calculus to show how replication is handled. Our proof exploits two main technical notions. The first idea is to introduce an induction principle on processes, that allows us to provide an inductive characterisation of $\simeq_{\text{int}}$. We then introduce modal formulas whose role is, intuitively, to establish that only finitely many terms have to be taken into consideration when exploring the outcomes of a given process.

Exploiting $\simeq_{\mathrm{int}}$, we relate logical equivalence with two important equivalences for processes. The first equivalence is the standard extensional equivalence, namely barbed congruence ($\approx$). Here the main result is that logical equivalence is strictly finer. As counterexamples to the inclusion $\approx \subseteq =_L$, we have found three axiom schemata. We do not know whether they are complete, that is, if they exactly describe the difference between the two relations on MA.

We then compare logical equivalence with a second relation, namely structural congruence ($\equiv$), an intensional and very discriminating equivalence. We establish an axiomatisation of logical equivalence on a rather broad class of processes, called $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ (defined in 5.1). The definition of $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ relies on an image-finiteness constraint that is lighter than the usual notion of image-finiteness in process calculi, because only certain subterms of processes are required to give rise to finitely many reducts. This subcalculus is shown to be Turing complete in Section 6. We are not aware of other axiomatisations of semantic equivalences (defined by operational, denotational, logical, or other means) in higher-order process calculi. Our result says that on $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, $=_L$ almost exactly coincides with structural congruence, the only difference being an 'eta law' for communication of the form mentioned above. This axiomatisation does not hold in the full MA, for instance because of the phenomenon of stuttering.

Communication in MA is asynchronous, in the sense that outputs have no continuation. We show in 5.2 that if asynchronous communication is dropped in favour of synchronous communication, then logical equivalence exactly coincides with structural congruence on the synchronous version of $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$.

The comparisons reveal the intensional flavour of AL. Although the logic has operators for looking into the parallel structure of processes, the intensionality of the logic was far from immediate, essentially for two reasons. The first reason is that not all syntactical constructions of MA are reflected in the logic, which entirely lacks operators for capabilities, communications, and replication. The second reason is that we adopt a weak interpretation for reductions (i.e., we abstract from actions internal to the processes); this makes it possible to handle infinite processes, but at the same time entails a loss of precision when describing properties of processes. In such a setting it is therefore surprising that $=_L$ is actually so close to $\equiv$, also because $\equiv$ is a very strong relation – a few axioms are the only difference with syntactic identity.

Being very close to a syntactical description of processes, the relation of structural congruence is decidable. As a consequence, in the subcalculus of MA where we show that $=_L$ coincides with $\equiv$, we can also derive decidability for $=_L$. However, the frontier with undecidability for $=_L$ is very subtle: we establish undecidability of $=_L$ in the full calculus by encoding the halting problem of a Turing machine. This boils down in our setting to specifing Turing machines in Mobile Ambients and building a scenario where the halting of a machine corresponds to the existence of *reduction loops*, i.e., of processes P, Q such that P reduces to Q and Q reduces to P. This encoding is a challenging 'programming task', since the process must return to its initial state modulo $=_L$; this is a demanding condition, since, as mentioned above, $=_L$ is a rather strong relation. For instance, one has to be very precise in garbage collecting dead code during the execution of the Turing machine.

Other related work Although not directly related from a technical point of view, a work worth mentioning is [15]. In that work, models of (enrichments of) relevant and linear logic are defined using Milner's SCCS. In particular, the interpretation of implication is reminscent of the definition of satisfaction for the guarantee operator ($\triangleright$) in AL. Dam however explicitely renounces giving sense to formulas that talk about the structure of processes, as is the case in the Ambient Logic.

As stated before, intensional bisimilarity is to AL what bisimilarity is to Hennessy-Milner logic. Approximants of intensional bisimilarity, that will be needed in our proofs of completeness, may also be expressed in terms of Ehrenfeucht-Fraïssé games for spatial logics, as shown in [16]. These equivalences are standard devices to establish expressiveness results. For instance, they have been exploited to obtain adjunct elimination properties of spatial logics in [6, 26].

This work is a revised and extended version of parts of [30] and [20], precisely, those parts that deal with issues related to separability of AL. A companion paper [21] studies expressiveness issues. By the time the writing of the present paper was completed, a few papers have appeared that make use of results or methods presented here. These are works that study the intensionality of spatial logics or decidability properties. Works related to the intensionality of spatial logics include [8] where the spatial logic is static, and [6, 5], where the logic is applied to reason on calculi that feature a simpler notion of space, with a strong interpretation of the temporal modality. A spatial logic for the π-calculus satisfying the property that logical equivalence coincides with behavioural equivalence has been studied in [19]. This logic is defined by removing modal operators like 0 or spatial conjunction, and keeping only 'contextual' operators (guarantee and revelation adjunct). A similar result, but for a logic that includes spatial conjunction and 0, has been established for a process calculus encompassing a form of distribution in [7]. Works related to the decidability properties of Mobile Ambients include [3, 27], that address questions of termination, and [2, 4], that consider reachability in *syntactic* subcalculi of MA (in the sense that these subcalculi are obtained by eliminating some syntactical constructs). It can be noted that our analysis of decidability (in Section 6) allows us to deduce a property in terms of reachability: as discussed above, we establish that one cannot detect the presence of *reduction loops* (i.e., the existence of processes P and Q that reduce to eachother). This in particular entails undecidability of reachability.

Structure of the paper. We define the Mobile Ambients calculus and the Ambient Logic in Section 2. Section 3 is devoted to the study of intensional bisimilarity, $\simeq_{\mathrm{int}}$. We show that $\simeq_{\mathrm{int}}$ is included in logical equivalence, $=_L$. Completeness, i.e., the reverse inclusion, is first proved only for finite MA processes. For this, we need a certain number of expressiveness results about AL from [21], which are collected in 3.3. The completeness proof for the whole calculus is presented in Section 4, which completes our study of $\simeq_{\mathrm{int}}$ by finally estabilishing that $\simeq_{\mathrm{int}}$ and $=_L$ coincide. The inductive characterisation of $\simeq_{\mathrm{int}}$ is given in 4.1, and the logical characterisation of the outcomes of a process in 4.3. We compare $=_L$ with barbed congruence and structural congruence in Section 5. The subcalculus $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, on which we establish an axiomatisation of $=_L$, is also introduced here. Subsection 5.2 explains how our results are modified when moving to synchronous Ambients. We present our encoding of Turing machines into $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ in Section 6, and give concluding remarks in Section 7.

2. Background

This section collects the necessary background for this paper. It includes the Mobile Ambients calculus [13] syntax and semantics, and the Ambient Logic [11].

2.1. **Syntax of Mobile Ambients.** We recall here the syntax of Mobile Ambients (MA) (we sometimes also call this calculus the Ambient calculus). In the calculus we study, only names, not capabilities, can be communicated; this allows us to work in an untyped calculus.

The calculus is asynchronous; a synchronous extension will be considered in Section 5. As in [11, 9, 10], the calculus has no restriction operator for creating new names.

Table 2.1 shows the syntax. Letters n, m, h range over names, x, y, z over variables; η ranges over names and variables. Both the set of names and the set of variables are infinite. The expressions $\mathsf{in}\,\eta$, $\mathsf{out}\,\eta$, and $\mathsf{open}\,\eta$ are the *capabilities.* Messages and abstractions are the *input/output* (I/O) primitives. A *guard* is either an abstraction or a capability. A process P is *single* iff there exists P' such that either $P \equiv \mathsf{cap}.P'$ for some cap or $P \equiv n[\,P'\,]$ for some n).

Abstraction is a binding construct, giving rise to the set of free variables of a process P, written fv(P). We ignore syntactic differences due to alpha conversion. We write fn(P) for the set of (free) names of process P. A *closed* process has no free variable. Unless explicitely stated, we use $P, Q, \ldots$ to range over *closed* processes in our definitions and results. Substitutions, ranged over with σ, are partial functions from variables to names. Given σ, we write $P\sigma$ to denote the result of the application of σ to P. Given two processes P and Q, we say that σ is a closing substitution for P and Q (in short, a closing substitution) if $P\sigma$ and $Q\sigma$ are closed processes. We also introduce another notation: $P\{^n\!/\!_x\}$ stands for the capture avoiding substitution of variable x with name n in P, and $P\{^n\!/\!_m\}$ stands for the process obtained by replacing name m with name n in P. Given n processes $P_1, \ldots, P_n$, we sometimes write $\Pi_{1\leq i\leq n}P_i$ for the parallel composition $P_1 \mid \ldots \mid P_n$.

Process contexts (simply called contexts) are processes containing an occurrence of a special process, called the hole. We use $\mathcal{C}$ to range over process contexts, and $\mathcal{C}\{\!|\ P\ |\!\}$ stands for the process obtained by replacing the hole in $\mathcal{C}$ with P. Given two processes P and Q, a *closing context for P and Q* (in short, a closing context) is a context $\mathcal{C}$ such that $\mathcal{C}\{\!|\ P\ |\!\}$ and $\mathcal{C}\{\!|\ Q\ |\!\}$ are closed processes.

$$
\begin{array}{llll}
h, k, \ldots n, m & & \textit{Names} & \\
x, y, \ldots & & \textit{Variables} & \\
\eta & & \textit{Names} \cup \textit{Variables} & \\
 & & & \\
 & & \textit{Capabilities} & \\
\mathsf{cap} ::= & \mathsf{in}\,\eta & \textit{(enter)} & \\
 \mid & \mathsf{out}\,\eta & \textit{(exit)} & \\
 \mid & \mathsf{open}\,\eta & \textit{(open)} &
\end{array}
\qquad
\begin{array}{rcll}
 & & & \textit{Processes} \\
P, Q, R & ::= & \mathbf{0} & \textit{(nil)} \\
 & \mid & P \mid Q & \textit{(parallel)} \\
 & \mid & !P & \textit{(replication)} \\
 & \mid & \mathsf{cap}.\,P & \textit{(prefixing)} \\
 & \mid & \eta[\,P\,] & \textit{(ambient)} \\
 & \mid & \{\eta\} & \textit{(message)} \\
 & \mid & (x)\,P & \textit{(abstraction)}
\end{array}
$$

Processes with the same internal structure are identified. This is expressed by means of the *structural congruence relation*, $\equiv$, the smallest congruence such that the following laws hold:

$$P \mid \mathbf{0} \;\equiv\; P \qquad P \mid Q \;\equiv\; Q \mid P \qquad P \mid (Q \mid R) \;\equiv\; (P \mid Q) \mid R$$

$$!P \;\equiv\; !P \mid P \qquad !\mathbf{0} \;\equiv\; \mathbf{0} \qquad !(P \mid Q) \;\equiv\; !P \mid !Q \qquad !!P \;\equiv\; !P$$

As a consequence of the results presented in [32], which works with a richer calculus than the one we study, we have:

$$\frac{}{\mathsf{open}\, n.\, P \mid n[\, Q\,] \longrightarrow P \mid Q}\ \texttt{Red-Open}$$

$$\frac{}{n[\, \mathsf{in}\, m.\, P_1 \mid P_2\,] \mid m[\, Q\,] \longrightarrow m[\, n[\, P_1 \mid P_2\,] \mid Q\,]}\ \texttt{Red-In}$$

$$\frac{}{m[\, n[\, \mathsf{out}\, m.\, P_1 \mid P_2\,] \mid Q\,] \longrightarrow n[\, P_1 \mid P_2\,] \mid m[\, Q\,]}\ \texttt{Red-Out}$$

$$\frac{}{\{\eta\} \mid (x)\, P \longrightarrow P\{\eta/x\}}\ \texttt{Red-Com} \qquad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}\ \texttt{Red-Par}$$

$$\frac{P \longrightarrow P'}{n[\, P\,] \longrightarrow n[\, P'\,]}\ \texttt{Red-Amb} \qquad \frac{P \equiv P' \qquad P' \longrightarrow P'' \qquad P'' \equiv P'''}{P \longrightarrow P'''}\ \texttt{Red-Str}$$

Table 1: The rules for reduction

Theorem 2.1. $\equiv$ *is decidable.*

Definition 2.2 (Finite process)**.** A process P is *finite* iff there exists a process P' with no occurrence of the replication operator such that $P \equiv P'$.

2.2. **Operational Semantics.** The semantics of the calculus is given by a reduction relation $\longrightarrow$. We shall sometimes use the phrase 'τ-transitions' to refer to $\longrightarrow$ transitions. The corresponding rules are given in Table 2.2. The reflexive and transitive closure of $\longrightarrow$ is written $\Longrightarrow$.

Behavioural equivalence is defined using reduction and observability predicates $\Downarrow_n$ that indicate whether a process can liberate an ambient named n: formally, $P \Downarrow_n$ holds if there are P', P'' such that $P \Longrightarrow n[\, P'\,] \mid P''$.

Definition 2.3 (barbed congruence, [29, 24])**.** A symmetric relation $\mathcal{R}$ between processes is a *barbed bisimulation* if $P \mathcal{R} Q$ implies:

(1) whenever $P \Longrightarrow P'$, there exists Q' such that $Q \Longrightarrow Q'$ and $P' \mathcal{R} Q'$;
(2) for each name n, $P \Downarrow_n$ iff $Q \Downarrow_n$.

Barbed bisimilarity, written $\dot{\approx}$, is the largest barbed bisimulation. Two processes P and Q are *barbed congruent*, written $P \approx Q$, if $\mathcal{C}\{\!|\, P \,|\!\} \ \dot{\approx}\ \mathcal{C}\{\!|\, Q \,|\!\}$ for all closing contexts $\mathcal{C}$.

2.3. **Ambient Logic.** The Ambient Logic (AL), is presented in Table 2). We use an infinite set of *logical variables*, ranged over with x, y, z; η ranges over names and variables. (We can use the same syntax as for variables and names of the Ambient calculus, since formula and process terms are separate.) We use $\mathcal{A}, \mathcal{B}, \ldots, \mathcal{F}, \mathcal{F}', \ldots$ to range over formulas.

The logic has the propositional connectives, $\top, \neg\mathcal{A}, \mathcal{A} \vee \mathcal{B}$, and universal quantification on names, $\forall x.\ \mathcal{A}$, with the standard logical interpretation. The temporal connective, $\Diamond\mathcal{A}$ is considered with a weak interpretation. The spatial connectives, 0, $\mathcal{A} \mid \mathcal{B}$, and $\eta[\,\mathcal{A}\,]$, are the logical counterpart of the corresponding constructions on processes. $\mathcal{A} \triangleright \mathcal{B}$ and $\mathcal{A}@\eta$

$\mathcal{A}$	$::=$	$\top$	*(true)*	classical logic
	$\mid$	$\neg\mathcal{A}$	*(negation)*	
	$\mid$	$\mathcal{A} \vee \mathcal{B}$	*(disjunction)*	
	$\mid$	$\forall\, x\, .\, \mathcal{A}$	*(universal quantification over names)*	
	$\mid$	$\Diamond\, \mathcal{A}$	*(sometime)*	temporal and spatial connectives
	$\mid$	0	*(void)*	
	$\mid$	$\eta[\, \mathcal{A}\,]$	*(edge)*	
	$\mid$	$\mathcal{A} \mid \mathcal{B}$	*(composition)*	
	$\mid$	$\mathcal{A} @ \eta$	*(localisation)*	logical adjuncts
	$\mid$	$\mathcal{A} \;\triangleright\; \mathcal{B}$	*(linear implication)*	

Table 2: The syntax of logical formulas

are the adjuncts of $\mathcal{A} \mid \mathcal{B}$ and $\eta[\, \mathcal{A}\,]$, in the sense of being, roughly, their inverse (see below). $\mathcal{A}\{n/x\}$ is the formula obtained from $\mathcal{A}$ by substituting variable x by name n. A formula without free variables is *closed*. Along the lines of the definition of process contexts, we define formula contexts as formulas containing an occurrence of a special *hole formula*.

We use $\mathcal{A}\{\!|\, \cdot\, |\!\}$ to range over formula contexts; then $\mathcal{A}\{\!|\, \mathcal{B}\, |\!\}$ stands for the formula obtained by replacing the hole in $\mathcal{A}\{\!|\, \cdot\, |\!\}$ with $\mathcal{B}$.

Definition 2.4 (Satisfaction)**.** The satisfaction relation is defined between closed processes and closed formulas as follows:

$$
\begin{array}{lcl}
P \models \top & \stackrel{\text{def}}{=} & \text{always true} \\
P \models \forall\, x\, .\, \mathcal{A} & \stackrel{\text{def}}{=} & \text{for any } n,\ P \models \mathcal{A}\{n/x\} \\
P \models \neg\, \mathcal{A} & \stackrel{\text{def}}{=} & \text{not } P \models \mathcal{A} \\
P \models \mathcal{A}_1 \mid \mathcal{A}_2 & \stackrel{\text{def}}{=} & \exists P_1, P_2 \text{ s.t. } P \equiv P_1 \mid P_2 \text{ and } P_i \models \mathcal{A}_i,\ i = 1, 2 \\
P \models \mathcal{A} \vee \mathcal{B} & \stackrel{\text{def}}{=} & P \models \mathcal{A} \text{ or } P \models \mathcal{B} \\
P \models n[\, \mathcal{A}\,] & \stackrel{\text{def}}{=} & \exists P' \text{ s.t. } P \equiv n[\, P'\,] \text{ and } P' \models \mathcal{A} \\
P \models 0 & \stackrel{\text{def}}{=} & P \equiv \mathbf{0} \\
P \models \Diamond \mathcal{A} & \stackrel{\text{def}}{=} & \exists P' \text{ s.t. } P \Longrightarrow P' \text{ and } P' \models \mathcal{A} \\
P \models \mathcal{A} @ n & \stackrel{\text{def}}{=} & n[\, P\,] \models \mathcal{A} \\
P \models \mathcal{A} \triangleright \mathcal{B} & \stackrel{\text{def}}{=} & \forall R,\ R \models \mathcal{A} \text{ implies } P \mid R \models \mathcal{B}
\end{array}
$$

The logic in [11] has also a *somewhere* connective, that holds of a process containing, at some arbitrary level of nesting of ambients, an ambient whose content satisfies $\mathcal{A}$. For the sake of simplicity, we omit this connective, but we believe that the addition of this connective would not change the results in the paper (in particular Theorem 3.29 can be adapted easily).

Lemma 2.5 ([11])**.** *If $P \equiv Q$ and $P \models \mathcal{A}$, then also $Q \models \mathcal{A}$.*

We give $\vee$ the least syntactic precedence, thus $\mathcal{A}_1 \rhd \mathcal{A}_2 \vee \mathcal{A}_3$ reads $(\mathcal{A}_1 \rhd \mathcal{A}_2) \vee \mathcal{A}_3$, and $\mathcal{A}_1 \rhd (\Diamond\mathcal{A}_2 \vee \Diamond\mathcal{A}_3)$ reads $\mathcal{A}_1 \rhd ((\Diamond\mathcal{A}_2) \vee (\Diamond\mathcal{A}_3))$. We shall use the following standard duals of disjunction and universal quantification:

$$\mathcal{A} \wedge B \stackrel{\text{def}}{=} \neg(\neg\mathcal{A} \vee \neg\mathcal{B}) \qquad\qquad \exists\, x \,.\, \mathcal{A} \stackrel{\text{def}}{=} \neg\forall\, x \,.\, \neg\mathcal{A}$$

Definition 2.6 (Logical equivalence)**.** For processes P and Q, we say that P and Q are *logically equivalent,* written $P =_L Q$, if for any closed formula $\mathcal{A}$ it holds that $P \models \mathcal{A}$ iff $Q \models \mathcal{A}$.

The remainder of this paper is devoted to the study of $=_L$ on MA and on some subcalculi of MA.

3. Intensional bisimilarity

In order to be able to carry out our programme for $=_L$, as discussed in the introduction, we look for a co-inductive characterisation of this relation, as a form of labelled bisimilarity. Before introducing the bisimilarity relation, we need to define labelled transitions on MA, and a few derived relations such as the *stuttering* relation.

3.1. Definitions.

3.1.1. *Labelled transitions and stuttering.*

Definition 3.1. Let P be a closed process. We write:

- $P \stackrel{\mathsf{cap}}{\to} P'$, where cap is a capability, if $P \equiv \mathsf{cap}.\, P_1 \mid P_2$ and $P' = P_1 \mid P_2$.
- $P \stackrel{\{n\}}{\to} P'$ if $P \equiv \{n\} \mid P'$.
- $P \stackrel{?n}{\to} P'$ if $P \equiv (x)\, P_1 \mid P_2$ and $P' \equiv P_1\{{}^{n}\!/\!{}_{x}\} \mid P_2$.
- $P \stackrel{\mu}{\Longrightarrow} P'$, where μ is one of the above labels, if $P \Longrightarrow \stackrel{\mu}{\to} \Longrightarrow P'$ (where $\Longrightarrow \stackrel{\mu}{\to} \Longrightarrow$ is relation composition).
- **(stuttering)** $P \xRightarrow{(M_1,M_2)^\star} P'$ if there is $i \geq 1$ and processes $P_1, \ldots, P_i$ with $P = P_1$ and $P' = P_i$ such that $P_r \stackrel{M_1}{\Longrightarrow} \stackrel{M_2}{\Longrightarrow} P_{r+1}$ for all $1 \leq r < i$.
- Finally, $\stackrel{\langle\mathsf{cap}\rangle}{\Longrightarrow}$ is a convenient notation for compacting statements involving capability transitions. $\stackrel{\langle\mathsf{in}\ \mathsf{n}\rangle}{\Longrightarrow}$ is $\xRightarrow{(\mathsf{out}\ n,\mathsf{in}\ n)^\star}$; similarly $\stackrel{\langle\mathsf{out}\ \mathsf{n}\rangle}{\Longrightarrow}$ is $\xRightarrow{(\mathsf{in}\ n,\mathsf{out}\ n)^\star}$; and $\stackrel{\langle\mathsf{open}\ \mathsf{n}\rangle}{\Longrightarrow}$ is $\Longrightarrow$.

We discuss in Example 3.3 below why stuttering is needed to capture logical equivalence in MA.

3.1.2. *Intensional bisimilarity,* $\simeq_{\text{int}}$. We present here our main labelled bisimilarity, *intensional bisimilarity*, written $\simeq_{\text{int}}$. This relation will be used to capture the separating power of $=_L$.

Intuitively, the definition of $\simeq_{\text{int}}$ is based on the observations made available by the logic either using built-in operators or through derived formulas for capabilities (see below).

Definition 3.2. A symmetric relation $\mathcal{R}$ on closed processes is an *intensional bisimulation* if $P\mathcal{R}Q$ implies:

(1) If $P \equiv P_1 \mid P_2$ then there are Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ and $P_i \mathcal{R} Q_i$, for $i = 1, 2$.
(2) If $P \equiv \mathbf{0}$ then $Q \equiv \mathbf{0}$.
(3) If $P \longrightarrow P'$ then there is Q' such that $Q \Longrightarrow Q'$ and $P' \mathcal{R} Q'$.
(4) If $P \stackrel{\mathsf{in}\, n}{\longrightarrow} P'$ then there is Q' such that $Q \stackrel{\mathsf{in}\, n}{\Longrightarrow} \xrightarrow{(\mathsf{out}\ n, \mathsf{in}\ n)^\star} Q'$ and $P' \mathcal{R} Q'$.
(5) If $P \stackrel{\mathsf{out}\, n}{\longrightarrow} P'$ then there is Q' such that $Q \stackrel{\mathsf{out}\, n}{\Longrightarrow} \xrightarrow{(\mathsf{in}\ n, \mathsf{out}\ n)^\star} Q'$ and $P' \mathcal{R} Q'$.
(6) If $P \stackrel{\mathsf{open}\, n}{\longrightarrow} P'$ then there is Q' such that $Q \stackrel{\mathsf{open}\, n}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$.
(7) If $P \stackrel{\{n\}}{\longrightarrow} P'$ then there is Q' such that $Q \stackrel{\{n\}}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$.
(8) If $P \stackrel{?n}{\longrightarrow} P'$ then there is Q' such that $Q \mid \{n\} \Longrightarrow Q'$ and $P' \mathcal{R} Q'$.
(9) If $P \equiv n[\, P' \,]$ then there is Q' such that $Q \equiv n[\, Q' \,]$ and $P' \mathcal{R} Q'$.

Intensional bisimilarity, written $\simeq_{\text{int}}$, is the largest intensional bisimulation. The definition of $\simeq_{\text{int}}$ induces a relation $\simeq^o_{\text{int}}$, defined on open terms by saying that $P \simeq^o_{\text{int}} Q$ iff for any closing substitution σ, $P\sigma \simeq_{\text{int}} Q\sigma$.

The definition of $\simeq_{\text{int}}$ has (at least) three intensional clauses, namely (1), (2) and (9), which allow us to observe parallel compositions, the terminated process, and ambients. These clauses correspond to the intensional connectives '$|$', '0' and '$n[\cdot]$' of the logic. The clause (8) for abstraction is similar to the input clause of bisimilarity in asynchronous message-passing calculi [1]. This is so because communication in MA is asynchronous (see also Subsection 5.2 below). Note that, using notation $\stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow}$ introduced above, items 4, 5, and 6 can be replaced by the following one:

- if $P \stackrel{\mathsf{cap}}{\to} P'$, then there is Q' such that $Q \stackrel{\mathsf{cap}}{\Longrightarrow} \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$.

As we have pointed out above, stuttering is used to capture some transitions of processes that the logic cannot detect. It gives rise to particular kinds of loops, that we illustrate in the following example.

Example 3.3 (Stuttering Loop)**.** Consider the processes

$$P \stackrel{\text{def}}{=} !\mathsf{open}\ n.\, \mathsf{in}\ n.\, \mathsf{out}\ n.\, \mathsf{in}\ n.\, \mathsf{out}\ n.\, n[\,\mathbf{0}\,] \mid n[\,\mathbf{0}\,]$$
$$Q \stackrel{\text{def}}{=} !\mathsf{open}\ n.\, \mathsf{in}\ n.\, \mathsf{out}\ n.\, \mathsf{in}\ n.\, \mathsf{out}\ n.\, n[\,\mathbf{0}\,] \mid \mathsf{in}\ n.\, \mathsf{out}\ n.\, n[\,\mathbf{0}\,]\,.$$

We have the following loop, modulo stuttering:

$$P \xrightarrow{(\mathsf{in}\ n, \mathsf{out}\ n)^\star} Q \xrightarrow{(\mathsf{in}\ n, \mathsf{out}\ n)^\star} P\ .$$

The existence of such pairs of processes that reduce one to each other modulo stuttering will play an important role in the axiomatization of $=_L$. We call such a situation a loop.

It holds that $P \not\simeq_{\text{int}} Q$; however, since $P \xrightarrow{(\mathsf{in}\ n, \mathsf{out}\ n)^\star} Q \xrightarrow{(\mathsf{in}\ n, \mathsf{out}\ n)^\star} P$, we have

$$\mathsf{out}\ n.\, P \ \simeq_{\text{int}}\ \mathsf{out}\ n.\, Q\,.$$

Actually, $\mathsf{out}\ n.\, P \ \approx\ \mathsf{out}\ n.\, Q.$, that is, these two processes are extensionally equivalent, and they are also equated by the logic (i.e., $\mathsf{out}\ n.\, P =_L \mathsf{out}\ n.\, Q$). But they would not be intensionally bisimilar without the stuttering relations.

The reason for this peculiarity is that, intuitively, these processes have the same behaviour in any testing context. To see why the extra capabilities of Q do not affect its behaviour, consider a reduction involving $\mathsf{out}\ n.\, P$, of the following shape:

$$n[\, m[\, \mathsf{out}\ n.\, P \mid R\,]\,] \longrightarrow n[\,\mathbf{0}\,] \mid m[\, P \mid R\,]\,.$$

Process $\mathsf{out}\ n.\,Q$ can match this transition using three reductions:

$$\begin{array}{rcl} n[\,m[\,\mathsf{out}\ n.\,Q \mid R\,]\,] & \longrightarrow & n[\,\mathbf{0}\,] \mid m[\,\mathsf{in}\ n.\,\mathsf{out}\ n.\,n[\mathbf{0}] \mid Q' \mid R\,] \\ & \longrightarrow & n[\,m[\,\mathsf{out}\ n.\,n[\mathbf{0}] \mid Q' \mid R\,]\,] \\ & \longrightarrow & n[\,\mathbf{0}\,] \mid m[\,P \mid R\,]\,, \end{array}$$

where Q' is $!\mathsf{open}\ n.\,\mathsf{in}\ n.\,\mathsf{out}\ n.\,\mathsf{in}\ n.\,\mathsf{out}\ n.\,n[\,\mathbf{0}\,]$. Conversely, the process $\mathsf{out}\ n.\,Q$ may be involved in the following scenario:

$$n[\,m[\,\mathsf{out}\ n.\,Q \mid R\,]\,] \longrightarrow n[\,\mathbf{0}\,] \mid m[\,Q \mid R\,]\,,$$

and the process $\mathsf{out}\ n.\,P$ can mimic this reduction.

If we set $Q' = !\mathsf{open}\ n.\,\mathsf{in}\ n.\,\mathsf{out}\ n.\,\mathsf{in}\ n.\,\mathsf{out}\ n.\,n[\mathbf{0}]$, we have

$$\begin{array}{rcl} n[\,m[\,\mathsf{out}\ n.\,P \mid R\,]\,] & \longrightarrow & n[\,\mathbf{0}\,] \mid m[\,n[\,\mathbf{0}\,] \mid Q' \mid R\,] \\ & \longrightarrow & n[\,\mathbf{0}\,] \mid m[\,Q' \mid \mathsf{in}\ n.\,\mathsf{out}\ n.\,\mathsf{in}\ n.\,\mathsf{out}\ n.\,n[\mathbf{0}] \mid R\,] \\ & \longrightarrow & n[\,m[\,Q' \mid \mathsf{out}\ n.\,\mathsf{in}\ n.\,\mathsf{out}\ n.\,n[\mathbf{0}] \mid R\,]\,] \\ & \longrightarrow & n[\,\mathbf{0}\,] \mid m[\,Q \mid R\,]\,. \end{array}$$

By contrast, stuttering does not show up in Safe Ambients [24], where movements are achieved by means of synchronisations between a capability and a *co-capability*, and alike models.

The following result is an easy consequence of the definition of $\simeq_{\mathrm{int}}$:

Lemma 3.4. $\simeq_{\mathrm{int}}$ *is an equivalence relation.*

Proof. The only point worth mentioning is that, for transitivity, to handle clause (8), one first needs to prove that $\simeq_{\mathrm{int}}$ is preserved by parallel compositions with messages (which is anyhow straightforward). □

However, it is not obvious that $\simeq_{\mathrm{int}}$ is preserved by all operators of the calculus, due to the fact that $\simeq_{\mathrm{int}}$ is, intrinsically, higher-order. Formally, $\simeq_{\mathrm{int}}$ is not higher-order, in that the labels of actions do not contain terms. Clause (3) of Definition 3.2, however, involves some higher-order computation, for a reduction may involve movement of terms (for instance, if the reduction uses rules `Red-In` or `Red-Out`). This, as usual in higher-order forms of bisimilarity, complicates the proof that bisimilarity is preserved by parallel composition.

3.2. **Congruence.** In this section, we establish congruence of intensional bisimilarity, using an auxiliary relation.

3.2.1. *Syntactical relation,* $\approxeq$. Our proof of congruence makes use of a second bisimilarity, $\approxeq$, that, by construction, is preserved by all operators of the calculus, and that is defined as follows:

Definition 3.5. A symmetric relation on processes $\mathcal{R}$ is a *syntax-based intensional bisimulation* if $P\mathcal{R}Q$ implies:

(1) If $P \equiv P_1 \mid P_2$ then there are Q_s ($s = 1, 2$) such that $Q \equiv Q_1 \mid Q_2$ and for all s $P_s\mathcal{R}Q_s$.
(2) If $P \equiv \mathsf{cap}.\,P'$ then there are Q', Q'' such that
    (a) $Q \equiv \mathsf{cap}.\,Q'$,

(b) $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q''$, and
(c) $P' \mathcal{R} Q'$.
(3) If $P \equiv \{n\}$ then $Q \equiv \{n\}$.
(4) If $P \equiv (x)\, P'$ then there is Q' such that
(a) $Q \equiv (x)\, Q'$ and
(b) for all n there is Q'' such that $\{n\} \mid Q \Longrightarrow Q''$ and $P'\{n/x\} \mathcal{R} Q''$.
(5) If $P \equiv n[\, P'\,]$ then there is Q' such that $Q \equiv n[\, Q'\,]$ and $P' \mathcal{R} Q'$.

$\approxeq$ is the largest syntax-based intensional bisimulation. Given two open terms P and Q, we say that $P \approxeq^o Q$ holds iff for any closing substitution σ, $P\sigma \approxeq Q\sigma$.

Clause (4) is typical of asynchronous calculi, as in clause (8) of Definition 3.2. The differences between the definitions of $\simeq_{\text{int}}$ and $\approxeq$ are the following. First, labelled transitions are replaced by structural congruence in the hypothesis of the corresponding clause. Second, clause (3) about reductions of related processes is removed. Note that a clause for the process $\mathbf{0}$ is not necessary (see Lemma 3.9 below).

Transitivity of $\approxeq$ is not obvious, because it is not immediate that $\approxeq$ is preserved under reductions (there is no clause for matching τ-transitions, and reductions (i.e., relation $\Longrightarrow$) are used in a few places, such as the stuttering relation in the clauses for movement.

We shall prove that $\simeq_{\text{int}}$ and $\approxeq$ coincide (Corollary 3.18 below). Thus, transitivity of $\approxeq$ will hold because of $\simeq_{\text{int}}$'s transitivity, and conversely, congruence of $\approxeq$ will ensure congruence of $\simeq_{\text{int}}$. This proof method, which exploits an auxiliary relation that is manifestly preserved by the operators of the calculus but that is not manifestly preserved under reductions, brings to mind Howe's proof technique for proving congruence of bisimilarity in higher-order languages [23]. In our case, however, the problem is simpler because of the intensional clauses (1) and (2) of the bisimilarity and because MA is not a fully higher-order calculus: terms may move during a computation, but they may not be copied as a consequence of a movement. We may say that MA is a *linear* higher-order calculus (indeed the congruence of $\simeq_{\text{int}}$ could also be proved directly, with a little more work).

In order to establish congruence of $\approxeq$, we introduce an important equality between processes, that plays a technical role here but will also be used when characterising logical equivalence in Section 5.

Definition 3.6 (Eta law, $\equiv_E$)**.** The *eta law* is given by the following equation:

$$(x)\,((x)\,P \mid \{x\}) = (x)\,P\,.$$

We use the eta law to define the following three relations:

- $\longrightarrow_\eta$ is the eta law oriented from left to right; that is, $P \longrightarrow_\eta Q$ holds if Q is obtained from P by applying the eta law once, from left to right, to one of its subterms (modulo $\equiv$).
- $\longrightarrow_\eta^*$ stands for the reflexive, transitive closure of $\longrightarrow_\eta$;
- $\equiv_E$ is the smallest congruence satisfying the laws of $\equiv$ plus the eta law.

In the lemma below, we write $P \longrightarrow_{\eta\mathtt{h}} P'$ if $P \longrightarrow_\eta P'$ and this represents a top-level rewrite step, i.e., we do not rewrite under capabilities and input prefixes. Similarly, $\longrightarrow_{\eta\mathtt{h}}^*$ is the reflexive and transitive closure of $\longrightarrow_{\eta\mathtt{h}}$.

Lemma 3.7. *Let $\mathcal{R}$ stand for $\longrightarrow_\eta$ or $\longrightarrow_{\eta\mathtt{h}}$. We say that*

(1) *$\mathcal{R}$ is confluent up to $\equiv$, that is, for all P, Q, R such that $P\mathcal{R}^*Q$ and $P\mathcal{R}^*R$, there is Q', R' such that $Q\mathcal{R}^*Q'$, $R\mathcal{R}^*R'$ and $Q' \equiv R'$.*

(2) $\mathcal{R}$ *is terminating, that is* $\mathcal{R}^*$ *is a well-founded order.*

We call the *eta normal form* of P (the *head eta normal form* of P, respectively) the unique normal form, up to $\equiv$, of $\longrightarrow_\eta$ (of $\longrightarrow_{\eta\mathtt{h}}$, respectively).

Remark 3.8 (Eta law and stuttering)**.** The eta law expresses a form of stuttering (in communication, as opposed to stuttering in movements – see Definition 3.1). The logic being insensitive to both forms of stuttering, we have to reason modulo the eta law.

We now present some results that are needed to prove congruence of $\approxeq$.

Lemma 3.9. *If* $\mathbf{0} \approxeq Q$ *then* $Q \equiv \mathbf{0}$.

Proof. Suppose $Q \equiv \mathbf{0}$ does not hold. This means that there exists Q', Q'' s.t. $Q \equiv Q' \mid Q''$, with Q' is of the form $(x)\ R$, $\{p\}$, $M.R$, or $n[\,R\,]$. Then by applying the corresponding clause in the definition of $\approxeq$, we deduce $Q \not\approxeq \mathbf{0}$, i.e., a contradiction. □

Lemma 3.10. $\equiv_E \subseteq \approxeq$ *and* $\equiv_E\approxeq\equiv_E \subseteq \approxeq$*.*

Proof. Straightforward from the definition of $\approxeq$. □

If $\mathcal{R}$ is a binary relation on processes, we note $\mathcal{R}\{{}^n\!/\!_m\}$ for the relation defined as $\{(P\{{}^n\!/\!_m\}, Q\{{}^n\!/\!_m\}).\ (P,Q) \in \mathcal{R}\}$.

Lemma 3.11. *If* $\mathcal{R}$ *is a* $\approxeq$*-bisimulation, then for any* n, m, $\mathcal{R}\{{}^n\!/\!_m\}$ *is a* $\approxeq$*-bisimulation.*

Proof. Since τ transitions are not tested in $\approxeq$, substitution is not mentioned in Def. 3.5. All clauses of the latter definition are obviously stable by substitution. □

Lemma 3.12. *For any possibly open processes* P *and* Q*, if* $P \approxeq^o Q$ *then* $\mathcal{C}\{\!|\ P\ |\!\} \approxeq^o \mathcal{C}\{\!|\ Q\ |\!\}$*, for all contexts* $\mathcal{C}$*.*

Proof. By induction on $\mathcal{C}$, using the definition of $\approxeq$. □

To prove that $\simeq_{\text{int}}$ and $\approxeq$ coincide, the main result we need is that $\approxeq$ is preserved under reductions:

Lemma 3.13. *Suppose* $P \approxeq Q$ *and* $P \longrightarrow P'$*. Then there is* Q' *such that* $Q \Longrightarrow Q'$ *and* $P' \approxeq Q'$*.*

Proof. By induction on the depth of the derivation proof of $P \longrightarrow P'$. We proceed by case analysis on the last rule used in the derivation.

- Rule `Red-struct`:
$$\frac{P \equiv P_1 \qquad P_1 \longrightarrow P_2 \qquad P_2 \equiv P_3}{P \longrightarrow P_3}$$
By Lemma 3.10, $P_1 \approxeq Q$; by induction $Q \Longrightarrow Q' \approxeq P_2$; again by Lemma 3.10, $Q' \approxeq P_3$.
- Rule `Red-Par`:
$$\frac{P_1 \longrightarrow P_1'}{P_1 \mid P_2 \longrightarrow P_1' \mid P_2}$$
By definition of $\approxeq$ there are Q_i such that $Q \equiv Q_1 \mid Q_2$ and $P_i \approxeq Q_i$. Then we conclude, using induction and Lemma 3.12.
- Rule `Red-Amb`: Use induction and Lemma 3.12.

- Rule `Red-Com`: Immediate by clauses (1), (3), and (4) of Definition 3.5.
- Rule `Red-Open`:

$$\mathsf{open}\, n.\, P_1 \mid n[\, P_2\,] \longrightarrow P_1 \mid P_2$$

  By definition of $\approxeq$, $Q \equiv \mathsf{open}\, n.\, Q_1 \mid n[\, Q_2\,]$, and for some Q_1' with $Q_1 \Longrightarrow Q_1'$, we have: $P_2 \approxeq Q_2$, $P_1 \approxeq Q_1'$. We also have $Q \Longrightarrow Q_1' \mid Q_2$. Using Lemma 3.12, we derive $P_1 \mid P_2 \approxeq Q_1' \mid Q_2$, which concludes the case.
- Rule `Red-In`:

$$n[\, \mathsf{in}\, m.\, P_1 \mid P_2\,] \mid m[\, P_3\,] \longrightarrow m[\, n[\, P_1 \mid P_2\,] \mid P_3\,]$$

  By definition of $\approxeq$, $Q \equiv n[\, \mathsf{in}\, m.\, Q_1 \mid Q_2\,] \mid m[\, Q_3\,]$, and there exists Q_1' such that $Q_1 \xRightarrow{(\mathsf{out}\ n, \mathsf{in}\ n)^\star} Q_1'$ and we have: $P_2 \approxeq Q_2$, $P_3 \approxeq Q_3$, and $P_1 \approxeq Q_1'$.

  We also have $Q \Longrightarrow m[\, n[\, Q_1' \mid Q_2\,] \mid Q_3\,]$. Using Lemma 3.12, we derive

$$m[\, n[\, P_1 \mid P_2\,] \mid P_3\,] \approxeq m[\, n[\, Q_1' \mid Q_2\,] \mid Q_3\,],$$

  which concludes the case.
- Rule `Red-Out`: similar to the previous case. □

Corollary 3.14. *Suppose $P \approxeq Q$ and $P \Longrightarrow P'$. Then there is Q' such that $Q \Longrightarrow Q'$ and $P' \approxeq Q'$.*

Proof. By induction on the number of transitions in $P \Longrightarrow P'$, using Lemma 3.13 for the inductive case. □

Lemma 3.15.
- *$\mathsf{cap}.\, P \simeq_{\mathrm{int}} Q$ implies $Q \equiv \mathsf{cap}.\, Q'$, for some Q'.*
- *$\{n\} \simeq_{\mathrm{int}} Q$ implies $Q \equiv \{n\}$.*
- *$(x)\, P \simeq_{\mathrm{int}} Q$ implies $Q \equiv (x)\, Q'$, for some Q'.*

Proof. In every case, we suppose by contadiction that $Q \equiv Q_1 \mid Q_2$ where none of the Q_is is structurally congruent to $\mathbf{0}$. Then P and Q can be distinguished using the clauses of $\simeq_{\mathrm{int}}$ for parallel composition and $\mathbf{0}$, which means a contradiction.

Therefore, Q is single (it has only one component), and we can conclude using the appropriate clause of the definition of $\simeq_{\mathrm{int}}$ in each case. □

Lemma 3.16. $\simeq_{\mathrm{int}} \subseteq \approxeq$.

Proof. By proving that $\simeq_{\mathrm{int}}$ is a $\approxeq$-bisimulation. The proof is easy, using Lemma 3.15. □

Lemma 3.17. $\approxeq \subseteq \simeq_{\mathrm{int}}$.

Proof. By proving that $\approxeq$ is a $\simeq_{\mathrm{int}}$-bisimulation. We need Lemma 3.12 (precisely, the fact that $\approxeq$ is preserved by parallel composition), Lemma 3.10, Corollary 3.14, and Lemma 3.9. □

Corollary 3.18. *Relations $\simeq_{\mathrm{int}}$ and $\approxeq$ coincide.*

Corollary 3.19. *Relations $\simeq_{\mathrm{int}}^{o}$ and $\approxeq^{o}$ are congruence relations.*

Proof. Follows from Corollary 3.18, and Lemmas 3.4 and 3.12 □

3.3. **Expressiveness results.** In this subsection we recall some expressiveness results for AL. These results state the existence of formulas capturing some nontrivial properties of processes. They are proved in [21], and will be exploited later to assess the separating power of the logic.

We start by introducing two measures on terms, that represent two ways of defining the *depth* of a process. The first definition exploits the notion of eta normal form (see Lemma 3.7):

Definition 3.20 (Sequentiality degree, sd)**.** The sequentiality degree of a term P is defined as follows:

- $\mathsf{sd}(\mathbf{0}) = 0$, $\mathsf{sd}(P \mid Q) = \max\big(\mathsf{sd}(P), \mathsf{sd}(Q)\big)$;
- $\mathsf{sd}(n[\,P\,]) = \mathsf{sd}(!P) = \mathsf{sd}(P)$;
- $\mathsf{sd}(\mathsf{cap}.\,P) = 1 + \mathsf{sd}(P)$;
- $\mathsf{sd}(\{n\}) = 1$;
- $\mathsf{sd}((x)\ P) = \mathsf{sd}(P') + 1$ where $(x)\ P'$ is the eta normal form of $(x)\ P$.

Intuitively, the sequentiality degree counts the number of 'parcels of interaction' (capabilities, messages, input prefixes) in a term. We now define the *depth degree*, that is sensitive to the number of nested ambients. This quantity will be soon used in the interpretation of some formulas of AL, but also to define an inductive order on processes (see Subsection 3.4).

Definition 3.21 (Depth degree)**.** The depth degree of a process is computed using a function dd from MA processes to natural numbers, inductively defined by:

- $\mathsf{dd}(\mathbf{0}) \stackrel{\text{def}}{=} 0$, $\mathsf{dd}(\mathsf{cap}.\,P) \stackrel{\text{def}}{=} 0$;
- $\mathsf{dd}((x)\ P) \stackrel{\text{def}}{=} 0$, $\mathsf{dd}(\{n\}) \stackrel{\text{def}}{=} 0$;
- $\mathsf{dd}(n[\,P\,]) \stackrel{\text{def}}{=} \mathsf{dd}(P) + 1$;
- $\mathsf{dd}((!)P_1 \mid \ldots \mid (!)P_r) \stackrel{\text{def}}{=} \max_{1 \le i \le r} \mathsf{dd}(P_i)$.

We introduce formulas that express some kind of *possibility modalities* corresponding to the movement capabilities and input prefix of MA.

Lemma 3.22. *For any* cap*, there exists a formula context* $\langle\langle \mathsf{cap} \rangle\rangle.\,\{\!|\ \cdot\ |\!\}$ *such that for any closed process* P*, and any formula* $\mathcal{A}$*,*

$$P \models \langle\langle \mathsf{cap} \rangle\rangle.\,\{\!|\ \mathcal{A}\ |\!\} \qquad \textit{iff} \qquad \exists P', P''.\ P \equiv \mathsf{cap}.\,P'\ ,\ P' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} P'' \ \textit{and}\ P'' \models \mathcal{A}\,.$$

For all n*, there is a formula* $\{n\}$ *such that*

$$P \models \{n\} \qquad \textit{iff} \qquad P \equiv \{n\}\,.$$

For all n*, there exists a formula context* $\langle\langle ?n \rangle\rangle.\,\{\!|\ \cdot\ |\!\}$ *such that for all process* P *and formula* $\mathcal{A}$*,*

$$P \models \langle\langle ?n \rangle\rangle.\,\{\!|\ \mathcal{A}\ |\!\} \qquad \textit{iff} \qquad \exists x, P', P''.\ P \equiv (x)P'\ ,\ (x)P' \mid \{n\} \Longrightarrow P'' \ \textit{and}\ P'' \models \mathcal{A}\,.$$

We will also need the *necessity modalities*, that have a dual interpretation w.r.t. the above formulas:

Lemma 3.23. *For all* cap*, there is a formula context* $[\![\mathsf{cap}]\!].\,\{\!|\ \cdot\ |\!\}$ *such that for all process* P *and formula* $\mathcal{A}$*,*

$$P \models [\![\mathsf{cap}]\!].\,\{\!|\ \mathcal{A}\ |\!\} \qquad \textit{iff} \qquad \exists P'.\ P \equiv \mathsf{cap}.\,P' \ \textit{and}\ \forall P''.\ P' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} P'' \ \textit{implies}\ P'' \models \mathcal{A}\,.$$

For all n, there is a formula context $[\![?n]\!].\{\!|\cdot|\!\}$ such that, for all process P and formula $\mathcal{A}$,

$$P \models [\![?n]\!].\{\!|\mathcal{A}|\!\} \quad \textit{iff} \quad \exists P', x.\ P \equiv (x)P' \textit{ and } \forall P''.\,(x)P' \mid \{n\} \Longrightarrow P'' \textit{ implies } P'' \models \mathcal{A}\,.$$

Each operator of the syntax of MA (Table 2.1) has thus a counterpart in the logic, except replication. It is possible to express in AL a restricted form of replication on formulas, by defining a formula $!\mathcal{A}$, expressing that there are infinitely many processes in parallel satisfying $\mathcal{A}$, modulo some additional condition on $\mathcal{A}$. More precisely, based on Definitions 3.20 and 3.21 above, we say that a formula $\mathcal{A}$ is *sequentially selective* (resp. *depth selective*) if all processes satisfying $\mathcal{A}$ have the same sequentiality degree (resp. depth degree).

Lemma 3.24. *For all cap, there exists a formula context $\mathsf{Rep}_{\mathsf{cap}}\{\!|\cdot|\!\}$ such that for all process P and for all sequentially selective formula $\mathcal{A}$, whose models are only of the form $\mathsf{cap}.\,R$,*

$$P \models \mathsf{Rep}_{\mathsf{cap}}\{\!|\mathcal{A}|\!\} \quad \textit{iff} \quad \exists P_1,\dots,P_r.\ P \equiv\, !P_1 \mid (!)P_2 \mid \dots \mid (!)P_r \textit{ and, } P_i \models \mathcal{A}\,, i = 1\dots r\,.$$

For all n, there is a formula $!\{n\}$ such that

$$P \models\, !\{n\} \quad \textit{iff} \quad P \equiv\, !\{n\}\ .$$

There exists a formula context $\mathsf{Rep}_{input}\{\!|\cdot|\!\}$ such that for all process P and for all formula $\mathcal{A}$ sequentially selective whose models are only of the form $(x)P$,

$$P \models \mathsf{Rep}_{input}\{\!|\mathcal{A}|\!\} \quad \textit{iff} \quad \exists P_1,\dots,P_r.\ P \equiv\, !P_1 \mid (!)P_2 \mid \dots \mid (!)P_r \textit{ and, } P_i \models \mathcal{A}\,, i = 1\dots r\,.$$

Similar results hold for the replicated version of the *dual* modalities. The notion of depth selectiveness allows us to derive formulas that capture replicated ambients:

Lemma 3.25. *For all n, there is a formula context $!n[\{\!|\cdot|\!\}]$ such that for all process P and for all depth selective formula $\mathcal{A}$,*

$$P \models\, !n[\{\!|\mathcal{A}|\!\}] \quad \textit{iff} \quad \exists P_1,\dots,P_r.\ P \equiv\, !P_1 \mid (!)P_2 \mid \dots \mid (!)P_r \textit{ and, } P_i \models n[\mathcal{A}]\,, i = 1\dots r\,.$$

By putting together these expressiveness results, we can derive formulas characterising the equivalence class of a process w.r.t. logical equivalence for a subcalculus of MA, defined as follows:

Definition 3.26 (Subcalculus $\mathrm{MA_{IF}}$)**.** Consider a process P, and a name $n \notin \mathrm{fn}(P)$. We say that P is *image-finite* if any subterm of P of the form $\mathsf{cap}.\,P'$ (resp. $(x)P'$) is such that the set

$$\{P'' \ :\ P' \stackrel{\langle\mathsf{cap}\rangle}{\Longrightarrow} P''\}_{/\simeq_{\mathrm{int}}}$$

(resp. $\{P'' \ :\ P'\{^n\!/\!_x\} \Longrightarrow P''\}_{/\simeq_{\mathrm{int}}}$) is finite. $\mathrm{MA_{IF}}$ is the set of image-finite MA processes.

In the standard definition of image-finiteness, as used, e.g., to establish inductively completeness of the Hennessy-Milner logic, one requires that the set of outcomes of *the process* is finite. While exploring the possible outcomes (and in absence of restriction in the process calculus), we may expose at top-level any subterm of the process, and hence we implicitly require that all of its subterms are image-finite in the standard sense. On the other hand, in our case, we do not impose that P has only finitely many outcomes, but only do so for *some* subterms. As a consequence, our notion is less restrictive, and any image-finite process in the standard sense belongs to $\mathrm{MA_{IF}}$.

Lemma 3.27 (Characteristic formulas on $\mathrm{MA_{IF}}$)**.** *For any closed $\mathrm{MA_{IF}}$ process P, there exists a formula $\mathcal{A}_P$ s.t. for any Q, these three conditions are equivalent:*

(1) $Q \models \mathcal{A}_P$;
(2) $P =_L Q$;
(3) $P \simeq_{\text{int}} Q$.

A final expressiveness result that will be needed later is the ability to test free name occurrences in a process.

Lemma 3.28. *For any name n, there exists a formula ©n such that for any P, $P \models$ ©n iff $n \in \text{fn}(P)$.*

3.4. **Soundness, and Completeness for Finite Processes.** We now study soundness and completeness of $\simeq_{\text{int}}$ with respect to $=_L$. Soundness means that $\simeq_{\text{int}} \subseteq\ =_L$, and completeness is the converse. We show here soundness on the whole calculus. By contrast, we only prove completeness on the finite processes, deferring the general result to the next section. We chose to do this for the sake of clarity: the proof in the finite case is much simpler, and exposes the basic ideas of the argument in the full calculus.

3.4.1. *Soundness on full public MA.* In order to prove soundness (on the whole calculus), we use the definition of $\cong$ and the congruence property to establish that bisimilar processes satisfy the same formulas.

Theorem 3.29 (Soundness of $\simeq_{\text{int}}$). *Assume $P, Q \in MA$, and suppose $P \simeq_{\text{int}} Q$. Then, for all $\mathcal{A}$, it holds that $P \models \mathcal{A}$ iff $Q \models \mathcal{A}$.*

Proof. By induction on the size of $\mathcal{A}$.

- $\mathcal{A} = \top$.
  Nothing to prove.
- $\mathcal{A} = \neg\,\mathcal{B}$ or $\mathcal{A} = \mathcal{B}_1 \vee \mathcal{B}_2$.
  By induction and the definition of satisfaction.
- $\mathcal{A} = 0$.
  By definition of satisfaction and clause (2) of the definition of $\simeq_{\text{int}}$.
- $\mathcal{A} = n[\,\mathcal{B}\,]$.
  Then $P \equiv n[\,P'\,]$ and $P' \models \mathcal{B}$. Hence $Q \equiv n[\,Q'\,]$ for some $Q' \simeq_{\text{int}} P'$. By induction, $Q' \models \mathcal{B}$; we can therefore conclude that also $Q \models n[\,\mathcal{B}\,]$ holds.
- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$.
  Then $P \equiv P_1 \mid P_2$ and $P_i \models \mathcal{A}_i$. By clause (1) of Definition 3.2, $Q \equiv Q_1 \mid Q_2$ for some $Q_i \simeq_{\text{int}} P_i$. By induction, $Q_i \models \mathcal{A}_i$; we can therefore conclude that also $Q \models \mathcal{A}_1 \mid \mathcal{A}_2$ holds.
- $\mathcal{A} = \forall\, x\, .\, \mathcal{B}$.
  By definition of satisfaction, $P \models \mathcal{B}\{n/x\}$ for all n. The result for Q then follows by induction, for $\mathcal{B}\{n/x\}$ is strictly small than $\forall\, x\, .\, \mathcal{B}$.
- $\mathcal{A} = \Diamond\, \mathcal{B}$.
  By definition of satisfaction, there is P' such that $P \Longrightarrow P'$ and $P' \models \mathcal{B}$. Using clause (3) of the definition of $\simeq_{\text{int}}$, there is Q' such that $Q \Longrightarrow Q' \simeq_{\text{int}} P'$. By induction, $Q' \models \mathcal{B}$; hence $Q \models \mathcal{A}$.
- $\mathcal{A} = \mathcal{B}@n$ or $\mathcal{A} = \mathcal{A}_1 \triangleright \mathcal{A}_2$.
  Follows using induction and the congruence of $\simeq_{\text{int}}$. $\square$

3.4.2. *Completeness, on finite processes.* The proof of completeness we develop here is based on the construction of a sequence of approximants of $\approxeq$, which is a standard approach for image-finite calculi. This works in the finite case (finiteness implies image-finiteness), but not in presence of replication. The proof is however interesting on its own, and gives a much simpler account on how the logic expresses the clauses of $\simeq_{\mathrm{int}}$ than the proof for the whole calculus.

Note that the definability of characteristic formulas for $\simeq_{\mathrm{int}}$ on $\mathrm{MA_{IF}}$ (see Definition 3.26 and Lemma 3.27) implies completeness: for two $\mathrm{MA_{IF}}$ processes P and Q, $P =_L Q$ entails $P \simeq_{\mathrm{int}} Q$. Since $\mathrm{MA_{IF}}$ contains the set of finite processes, this already gives completeness on finite processes. We nevertheless present here a proof that is specific to the finite case, to prepare the ground for completeness on full public MA. The route we are interested in for the completeness proof uses i-th approximants $\approxeq_i$ of relation $\approxeq$, and the fact that $\approxeq_\omega \stackrel{\mathrm{def}}{=} \bigcap_i \approxeq_i$ coincides with $\approxeq$.

Definition 3.30. We define the relations $\approxeq_i$ between processes, for all $i \geq 0$, as follows.

$\approxeq_0$ is the universal relation, and $\approxeq_{i+1}$ is defined by saying that $P \approxeq_{i+1} Q$ holds if we have:

(1) If $P \equiv P_1 \mid P_2$ then there are Q_s $(s = 1, 2)$ such that $Q \equiv Q_1 \mid Q_2$ and for all s $P_s \approxeq_i Q_s$.
(2) If $P \equiv \mathsf{cap}.\, P'$ then there are Q', Q'' such that
   (a) $Q \equiv \mathsf{cap}.\, Q'$,
   (b) $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q''$, and
   (c) $P' \approxeq_i Q'$.
(3) If $P \equiv \{n\}$ then $Q \equiv \{n\}$.
(4) If $P \equiv (x)\, P'$ then there is Q' such that
   (a) $Q \equiv (x)\, Q'$ and
   (b) for all n there is Q'' such that $\{n\} \mid Q \Longrightarrow Q''$ and $P'\{^n\!/\!_x\} \approxeq_i Q''$.
(5) If $P \equiv n[\, P'\,]$ then there is Q' such that $Q \equiv n[\, Q'\,]$ and $P' \approxeq_i Q'$.

We set $\approxeq_\omega \stackrel{\mathrm{def}}{=} \bigcap_{i \geq 0} \approxeq_i$.

Lemma 3.31. *$\approxeq_\omega$ coincides with $\approxeq$ on finite processes.*

Proof. Standard approximation result (finite processes are image finite). □

Lemma 3.32. *Let P, Q be two* finite *processes. If $P =_L Q$ then $P \approxeq_\omega Q$.*

Proof. Suppose $P \not\approxeq_\omega Q$. Then there is i such that $P \not\approxeq_i Q$. We prove, by induction on i, that in this case we can find a formula $\mathcal{A}$ such that $P \models \mathcal{A}$ holds but $Q \models \mathcal{A}$ does not.

For $i = 0$, this trivially holds since the hypothesis $P \not\approxeq_0 Q$ is absurd for $\approxeq_0$ being the universal relation.

Now the case $i + 1$, for $i \geq 0$. We proceed by case analysis:

(1) $P \equiv P_1 \mid P_2$, and for all Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ there is t $(1 \leq t \leq 2)$ such that $P_t \not\approxeq_i Q_t$.

   Modulo $\equiv$, there is a finite number, say s, of pairs of processes Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ (note that by hypothesis P is finite). Call $Q_{t,u}$ the t-th process of the u-th pair. Then for all u $(1 \leq u \leq s)$ there is t such that $P_t \not\approxeq_i Q_{t,u}$. By induction, there is $\mathcal{A}_{t,u}$ such that

$$P_t \models \mathcal{A}_{t,u} \qquad \text{and} \qquad Q_{t,u} \not\models \mathcal{A}_{t,u}\,.$$

Define

$$B_t \stackrel{\text{def}}{=} \bigwedge_{u.\,1\le u\le s \text{and } P_t \not\approx_i Q_{t,u}} \mathcal{A}_{t,u}\,.$$

Then

$$P \models B_1 \mid B_2\,,$$

whereas

$$Q \not\models B_1 \mid B_2\,.$$

(2) $P \equiv \mathsf{cap}.\,P'$; then necessarily $Q \equiv \mathsf{cap}.\,Q'$, and for all Q_t such that $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Longrightarrow} Q_t$, it holds that $P' \not\approx_i Q_t$.

By induction, for all t there is $\mathcal{A}_t$ such that $P' \models \mathcal{A}_t$ but $Q_t \not\models \mathcal{A}_t$. Since Q is finite, there is only a finite number of such processes Q_t (up to $\equiv$). Write $(Q_t)_{t\in I}$ for this set of processes up to $\equiv$ (we pick a representant for each $\equiv$-equivalence class), and call $\mathcal{A}_t$ the formula corresponding to each Q_t. Define

$$\mathcal{A} \stackrel{\text{def}}{=} \langle\langle\mathsf{cap}\rangle\rangle.\,\{\!| \bigwedge_{t\in I} \mathcal{A}_t |\!\}\,,$$

using the standard notation for the (finite) conjunction of the $\mathcal{A}_t$s. Then $P \models \mathcal{A}$ but $Q \not\models \mathcal{A}$.

(3) $P \equiv \{n\}$, and $Q \not\equiv \{n\}$: then $P{\models}\{n\}$, and $Q \not\models\{n\}$.

(4) $P \equiv (x)\,P'$, $Q \equiv (x)\,Q'$ and there is n such that for all Q_t such that $\{n\} \mid Q \Longrightarrow Q_t$, it holds that $P'' \not\approx_i Q_t$, for $P'' \stackrel{\text{def}}{=} P'\{{}^{n}\!/\!_{x}\}$.

Modulo $\equiv$, there is only a finite number of such Q_ts, say $Q_1,\ldots,Q_s$. By induction, there are formulas $\mathcal{A}_1,\ldots,\mathcal{A}_s$ with $P'' \models \mathcal{A}_t$ and $Q_t \not\models \mathcal{A}_t$. We introduce as above the notation $(Q_t)_{t\in I}$, and we define

$$\mathcal{A} \stackrel{\text{def}}{=} \langle\langle ?n\rangle\rangle.\,\{\!| \bigwedge_{t\in I} \mathcal{A}_t |\!\}\,.$$

Then $P \models \mathcal{A}$, but $Q \not\models \mathcal{A}$, because whenever $\{n\} \mid Q \Longrightarrow Q_t$, it holds that $Q_t \not\models \mathcal{A}_t$.

(5) $P \equiv n[\,P'\,]$, $Q \equiv n[\,Q'\,]$ and $P' \not\approx_i Q'$.

By induction there is $\mathcal{A}'$ with $P' \models \mathcal{A}'$ but $Q' \not\models \mathcal{A}'$. Define $\mathcal{A} \stackrel{\text{def}}{=} n[\,\mathcal{A}'\,]$; then $P \models \mathcal{A}$ but $Q \not\models \mathcal{A}$. □

Theorem 3.33 (Completeness on finite processes)**.** *Let P, Q be two finite closed processes. If $P =_L Q$ then $P \simeq_{\text{int}} Q$.*

Proof. Follows from Lemma 3.31 and 3.32. □

4. Completeness of $\simeq_{\text{int}}$ in the full calculus

The proof we have presented in the finite case cannot be used directly in the full MA calculus, because we lack the image-finiteness hypothesis, which allowed us to show that the limit $\approx_\omega$ coincides with $\approx$. In this section, we present a proof of the completeness of $\simeq_{\text{int}}$ for all processes. To do this, we establish the existence, for any processes P, Q, of a formula $\mathcal{F}_{P,Q}$ such that $P{\models}\mathcal{F}_{P,Q}$, and such that $Q{\models}\mathcal{F}_{P,Q}$ holds if and only if $P \simeq_{\text{int}} Q$. This result is hence weaker than the existence of characteristic formulas, but it does not require image finiteness.

We sketch the structure of the proof. Our approach exploits two technical devices, that we introduce first. We start by proving some lemmas related to the sequentiality degree of a term (Definition 3.20), which allows us to define a sound induction principle on MA processes. This principle supports the introduction of an inductive characterisation of $\simeq_{\text{int}}$. The second technical device we introduce is the set of *frozen subterms* of a process, that intuitively corresponds to the collection of subterms appearing under guards (capabilities or input prefixes) in a given term. These two technical notions are then used to define *local characteristic formulas*, which correspond to a relaxed notion of characteristic formula w.r.t. logical equivalence. An important fact about the set of frozen subterms of a process is that it enjoys a kind of subject reduction property; this allows us to replace the potentially infinite set of images of a term with a finite set when constructing local characteristic formulas.

4.1. **An inductive characterisation of $\simeq_{\text{int}}$.** We now establish some properties related to the sequentiality degree of processes. These allow us to introduce a well-founded order on terms which supports the definition of an inductive relation that coincides with $\simeq_{\text{int}}$.

Lemma 4.1. *Let P, Q be two terms of MA. Then:*
(1) if $P \equiv Q$, then $\mathsf{sd}(P) = \mathsf{sd}(Q)$;
(2) if $P \longrightarrow Q$ or $P \stackrel{\mu}{\to} Q$ then $\mathsf{sd}(P) \geq \mathsf{sd}(Q)$.

Proof. 1 is immediate, as is the result on $\stackrel{\mu}{\to}$ in 2. For $P \longrightarrow Q$, we reason by induction on the height of the derivation of $P \longrightarrow Q$. □

Corollary 4.2. *For all cap, if $P \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q$, then $\mathsf{sd}(P) \geq \mathsf{sd}(Q)$.*

This result will be important for the justification of Definition 4.9 below.

Lemma 4.3. *For any closed process $P \in MA$, there exists a formula $\mathcal{F}_{\mathsf{sd}(P)}$ such that:*
- *$P \models \mathcal{F}_{\mathsf{sd}(P)}$, and*
- *for any term Q, if $Q \models \mathcal{F}_{\mathsf{sd}(P)}$, then $\mathsf{sd}(Q) \geq \mathsf{sd}(P)$.*

Proof. We can assume that P is eta normalised. Let us first reason by induction on $\mathsf{sd}(P)$:
- for $\mathsf{sd}(P) = 0$, $\mathcal{F}_{\mathsf{sd}(P)} = \top$ is sufficient.
- for $\mathsf{sd}(P) > 0$, let us assume that there exist formulas $\mathcal{F}_{\mathsf{sd}(P')}$ for any P' such that $\mathsf{sd}(P') < \mathsf{sd}(P)$. We reason by induction on P.
  - the case $P = \mathbf{0}$ is impossible.
  - for $P = P_1 \mid P_2$, there is i such that $\mathsf{sd}(P) = \mathsf{sd}(P_i)$. Then we may choose $\mathcal{F}_{\mathsf{sd}(P)} = \mathcal{F}_{\mathsf{sd}(P_i)} \mid \top$. In the same way, let us set $\mathcal{F}_{\mathsf{sd}(\{n\})} = \mathcal{F}_{\{n\}}$, $\mathcal{F}_{\mathsf{sd}(!P)} = \mathcal{F}_{\mathsf{sd}(P)} \mid \top$ and $\mathcal{F}_{\mathsf{sd}(n[\,P\,])} = n[\,\mathcal{F}_{\mathsf{sd}(P)}\,]$.
  - for $P = \mathsf{cap}.\,P'$, we use the general induction hypothesis to construct $\mathcal{F}_{\mathsf{sd}(P')}$. Let us then take $\mathcal{F}_{\mathsf{sd}(P)} = \langle\langle \mathsf{cap} \rangle\rangle.\,\mathcal{F}_{\mathsf{sd}(P')}$. Then $P \models \mathcal{F}_{\mathsf{sd}(P)}$, and for any Q such that $Q \models \mathcal{F}_{\mathsf{sd}(P)}$, we deduce (from Lemma 3.22) that there are Q', Q'' such that $Q \equiv \mathsf{cap}.\,Q'$ and $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q''$ with $Q'' \models \mathcal{F}_{\mathsf{sd}(P')}$. Now by Lemma 4.1, $\mathsf{sd}(Q) - 1 = \mathsf{sd}(Q') \geq \mathsf{sd}(Q'')$, and by induction hypothesis $\mathsf{sd}(Q'') \geq \mathsf{sd}(P') = \mathsf{sd}(P) - 1$, so that finally $\mathsf{sd}(Q) \geq \mathsf{sd}(P)$.
  - for $P = (x)\,P'$, we use the general induction hypothesis to get $\mathcal{F}_{\mathsf{sd}(P')}$. Let us then take $\mathcal{F}_{\mathsf{sd}(P)} = \exists x.\ \langle\langle ?x \rangle\rangle.\,\mathcal{F}_{\mathsf{sd}(P')}$. Then $P \models \mathcal{F}_{\mathsf{sd}(P)}$, and for any Q such that $Q \models \mathcal{F}_{\mathsf{sd}(P)}$, we deduce (from Lemma 3.22) that there are n, Q', Q'' such that $Q \equiv (x)\,Q'$ and

$Q_1 = \{n\} \mid (x)Q' \Longrightarrow Q''$ with $Q'' \models \mathcal{F}_{\mathsf{sd}(P')}$. Now by Lemma 4.1, $\mathsf{sd}(Q_1) - 1 = \mathsf{sd}(Q) - 1 = \mathsf{sd}(Q'\{^n\!/\!x\}) \geq \mathsf{sd}(Q'')$, and by induction hypothesis $\mathsf{sd}(Q'') \geq \mathsf{sd}(P') = \mathsf{sd}(P) - 1$, so that finally $\mathsf{sd}(Q) \geq \mathsf{sd}(P)$. □

A similar result can be proved for the depth degree of a process:

Lemma 4.4. *For any closed process $P \in MA$, there exists a formula $\mathcal{F}_{\mathsf{dd}(P)}$ such that:*
- *$P \models \mathcal{F}_{\mathsf{dd}(P)}$, and*
- *for any term Q, if $Q \models \mathcal{F}_{\mathsf{dd}(P)}$, then $\mathsf{dd}(Q) \geq \mathsf{dd}(P)$.*

Proof. We reason as in the proof of the previous lemma. □

Corollary 4.5. *If $P \simeq_{\text{int}} Q$, then $\mathsf{sd}(P) = \mathsf{sd}(Q)$ and $\mathsf{dd}(P) = \mathsf{dd}(Q)$.*

Proof. By Theorem 3.29, $P \simeq_{\text{int}} Q$ implies $P =_L Q$, which gives the result. □

The sequentiality degree can be used as a basis for inductive reasoning on processes up to reductions of some subterms. This is formalized by the following definition:

Definition 4.6 (Well-founded order)**.** Given two processes P and Q , we write $P < Q$ (or $Q > P$) if either $\mathsf{sd}(P) < \mathsf{sd}(Q)$ or P is a strict subterm of Q.

Lemma 4.7.
- *$<$ is well-founded.*
- *Suppose P is of the form either $\mathsf{cap}.P'$ or $(x)P'$, and suppose moreover $P > Q$ and $Q \overset{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'$ for some cap. Then $P > Q'$.*

Proof. • Well-foundedness: if P is a strict subterm of Q, then $\mathsf{sd}(P) \leq \mathsf{sd}(Q)$.
• $P > Q'$: follows from Lemma 4.1. □

In order to give an inductive characterisation of $\simeq_{\text{int}}$, we establish the following results about $\simeq_{\text{int}}$. These are *inversion properties*, in the sense that they allow one to deduce, from $P \simeq_{\text{int}} Q$, with P having a given shape, consequences about the shape of Q.

Lemma 4.8 (Inversion results for $\simeq_{\text{int}}$)**.** *Let P, P_1, P_2, Q be processes of MA. Then*
(1) $\mathbf{0} \simeq_{\text{int}} Q$ *iff* $Q \equiv \mathbf{0}$.
(2) $n[\,P\,] \simeq_{\text{int}} Q$ *iff there exists Q' such that $Q \equiv n[\,Q'\,]$ and $P \simeq_{\text{int}} Q'$.*
(3) $P_1 \mid P_2 \simeq_{\text{int}} Q$ *iff there exist Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ and $P_i \simeq_{\text{int}} Q_i$ for $i = 1, 2$.*
(4) $!P \simeq_{\text{int}} Q$ *iff there exist $r \geq 1, s \geq r, Q_i$ $(1 \leq i \leq s)$ such that $Q \equiv \Pi_{1 \leq i \leq r}!Q_i \mid \Pi_{r+1 \leq i \leq s} Q_i$, and $P \simeq_{\text{int}} Q_i$ for $i = 1 \ldots s$.*
(5) $\mathsf{cap}.P \simeq_{\text{int}} Q$ *iff there exists Q' such that $Q \equiv \mathsf{cap}.Q'$ with $P \overset{\langle \mathsf{cap} \rangle}{\Longrightarrow} \simeq_{\text{int}} Q'$ and $Q' \overset{\langle \mathsf{cap} \rangle}{\Longrightarrow} \simeq_{\text{int}} P$.*
(6) $\{n\} \simeq_{\text{int}} Q$ *iff* $Q \equiv \{n\}$
(7) $(x)\,P \simeq_{\text{int}} Q$ *iff there exists $Q', m.$ such that $m \notin \text{fn}(P) \cup \text{fn}(Q)$, $Q \equiv (x)\,Q'$ $Q \mid \{m\} \Longrightarrow \simeq_{\text{int}} P\{^m\!/\!x\}$ and $(x)\,P \mid \{m\} \Longrightarrow \simeq_{\text{int}} Q'\{^m\!/\!x\}$.*

Proof. We first leave out the fourth case.

For the other cases, the left to right implications follow by the fact that, in each case, the corresponding clauses in the definitions of $\approxeq$ and $\simeq_{\text{int}}$ are almost the same.

For the right to left implication, cases 1 and 6 hold by reflexivity of $\simeq_{\text{int}}$, and cases 2 and 3 follow from congruence of $\simeq_{\text{int}}$ (Corollary 3.19). Case 5 is similar to the corresponding condition in $\approxeq$ (note that all other conditions are trivially fulfilled).

We explain case 7 in more details. We take P, Q, Q', x, m satisfying the required properties, and further introduce processes P_1 and Q_1 by imposing $(x)P \mid \{m\} \Longrightarrow P_1 \simeq_{\text{int}} Q'\{^m\!/\!_x\}$ and $Q \mid \{m\} \Longrightarrow Q_1 \simeq_{\text{int}} P\{^m\!/\!_x\}$. To show that $P \simeq_{\text{int}} Q$, we need to show that these processes satisfy the condition for receptions in the definition of $\approxeq$ (Definition 3.5), all other requirements being satisfied. Consider an arbitrary name m', we want to show that there exist P'', Q'' such that $Q \mid \{m'\} \Longrightarrow Q''$, $P\{^{m'}\!/\!_x\} \simeq_{\text{int}} Q''$, $(x)P \mid \{m'\} \Longrightarrow P''$ and $P'' \simeq_{\text{int}} Q'\{^{m'}\!/\!_x\}$. By hypothesis, this holds for $m' = m$, by taking $P'' = P_1$ and $Q'' = Q_1$. Otherwise, we set $P'' = P_1\{^{m'}\!/\!_m\}$ and $Q'' = Q_1\{^{m'}\!/\!_m\}$. Then $(x)P \mid \{m'\} = ((x)P \mid \{m\})\{^{m'}\!/\!_m\} \Longrightarrow P_1\{^{m'}\!/\!_m\} = P''$ since $\Longrightarrow$ is closed under name replacement, and $Q \mid \{m'\} \Longrightarrow Q''$ for the same reason. Moreover, since $\simeq_{\text{int}}$ is also closed under name replacement (Lemma 3.11), we deduce from the hypothesis $P_1 \simeq_{\text{int}} Q'\{^m\!/\!_x\}$ that $P'' \simeq_{\text{int}} Q'\{^{m'}\!/\!_x\}$, and similarly from $Q_1 \simeq_{\text{int}} P\{^m\!/\!_x\}$ that $Q'' \simeq_{\text{int}} P\{^{m'}\!/\!_x\}$. As a consequence, the condition is established for all m'. Note that the hypothesis about m being fresh for P, Q is crucial in the proof above.

We are thus left with case 4. The right to left implication holds because, if we define $\mathcal{R}$ as $\approxeq$ extended with all pairs of the form $(P, \Pi_{1\le i\le r}!Q_i \mid \Pi_{r+1\le i\le s}Q_i)$, with the above conditions, then $\mathcal{R}$ satisfies the clauses of $\approxeq$, hence $\mathcal{R}$ is $\approxeq$. We now consider the left to right implication. First, note that by applying clauses 1 and 2 of Def. 3.2, it can be shown that for any two bisimilar processes P, Q, if $P \equiv P' \mid P' \mid .. P' \mid P''$, where P contains at least n copies of some single process P', then necessarily $Q \equiv Q_1 \mid .. \mid Q_n \mid Q'$ with $Q_i \approxeq P'$ for all i. This entails the left to right implication in the case where P is a single process. When P is not single, we write $P \equiv \Pi_{1\le i\le r}!P_i \mid \Pi_{r+1\le i\le s}P_i$, where $P_1, .., P_s$ are single processes. Thanks to the congruence rule $!(R_1 \mid R_2) = !R_1 \mid !R_2$, $!P \equiv !P_1 \mid .. \mid !P_s$. Assume $!P \approxeq Q$. Applying the inversion rule for parallel composition, we have $Q \equiv Q_1 \mid Q_s$ with, for every i, $!P_i \approxeq Q_i$, that is, using our reasoning on single processes, $Q_i \equiv \Pi_{1\le j\le r_i}!Q_{i,j} \mid \Pi_{r_i+1\le j\le s_i}Q_{i,j}$. Using the law $!R \equiv !R \mid !R$, it is possible to choose all r_i equal, and similarly applying $!R \equiv !R \mid R$ we can choose all s_i equal. It is then a matter of rearranging the $Q_{i,j}$ in $Q'_1 \mid .. \mid Q'_s$ to write Q in the expected form. □

We can now define the inductively defined relation that characterises $\simeq_{\text{int}}$.

Definition 4.9. Let $\sim_{\text{ind}}$ be the binary relation $P \sim_{\text{ind}} Q$ defined by induction on P for the order $<$ as follows:

(1) $\mathbf{0} \sim_{\text{ind}} Q$ if $Q \equiv \mathbf{0}$.
(2) $n[\,P\,] \sim_{\text{ind}} Q$ if there exists Q' such that $Q \equiv n[\,Q'\,]$ and $P \sim_{\text{ind}} Q'$.
(3) $P_1 \mid P_2 \sim_{\text{ind}} Q$ if there exist Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ and $P_i \sim_{\text{ind}} Q_i$ for $i = 1, 2$.
(4) $!P \sim_{\text{ind}} Q$ if there exist $r \geq 1, s \geq r, Q_i$ $(1 \leq i \leq s)$ such that $Q \equiv \Pi_{1\le i\le r}!Q_i \mid \Pi_{r+1\le i\le s}Q_i$, and $P \sim_{\text{ind}} Q_i$ for $i = 1 \ldots s$.
(5) $\mathsf{cap}.\,P \sim_{\text{ind}} Q$ if there exists Q' such that $Q \equiv \mathsf{cap}.\,Q'$ with $P \stackrel{\langle\mathsf{cap}\rangle}{\Longrightarrow} \sim_{\text{ind}} Q'$ and $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Longrightarrow} \sim_{\text{ind}} P$.
(6) $\{n\} \sim_{\text{ind}} Q$ if $Q \equiv \{n\}$
(7) $(x)\,P \sim_{\text{ind}} Q$ if there exists $Q', m.$ such that $m \notin \text{fn}(P) \cup \text{fn}(Q)$, $Q \equiv (x)\,Q'$, $Q \mid \{m\} \Longrightarrow \sim_{\text{ind}} P\{^m\!/\!_x\}$ and $(x)\,P \mid \{m\} \Longrightarrow \sim_{\text{ind}} Q'\{^m\!/\!_x\}$.

Theorem 4.10. *Relation $\sim_{\text{ind}}$ is well defined. Moreover, relations $\sim_{\text{ind}}$ and $\simeq_{\text{int}}$ coincide.*

Proof. The definition of $\sim_{\text{ind}}$ is justified using Lemma 4.7. The inclusion $\simeq_{\text{int}} \subseteq \sim_{\text{ind}}$ is established using the results of Lemma 4.8, which correspond precisely to the defining clauses of $\sim_{\text{ind}}$. The converse inclusion follows from Lemma 4.8 too. □

4.2. **Frozen subterms.** We now introduce the notion of frozen subterms of a process. The frozen subterms of a process correspond to occurrences that do not participate in immediate interactions but that may play a role in future reductions.

In the reminder, we use N to range over sets of names. Unless otherwise stated, we always implicitly suppose that such a set is finite.

Definition 4.11 (Frozen subterms)**.** Let N be a set of names; the set $\mathsf{froz}_N(P)$ is defined by induction on P as follows:

- $\mathsf{froz}_N(\mathbf{0}) = \mathsf{froz}_N(\{n\}) = \emptyset$;
- $\mathsf{froz}_N(P_1 \mid P_2) = \mathsf{froz}_N(P_1) \cup \mathsf{froz}_N(P_2)$;
- $\mathsf{froz}_N(!P) = \mathsf{froz}_N(P)$;
- $\mathsf{froz}_N(\mathsf{cap}.\,P) = \{P\} \cup \mathsf{froz}_N(P)$;
- $\mathsf{froz}_N((x)P) = \bigcup_{n\in N}\{P\{^n\!/\!_x\}\} \cup \mathsf{froz}_N(P\{^n\!/\!_x\})$.

If P, P' are two structurally congruent terms, then, modulo $\equiv$, $\mathsf{froz}_N(P) = \mathsf{froz}_N(P')$. Hence this set (in its quotiented version with respect to $\equiv$) is uniquely determined by the structural congruence class of P.

Lemma 4.12 (Finiteness of $\mathsf{froz}_N(P)$)**.** *For any $P \in MA$, if N is finite, then the set obtained by taking the quotient of $\mathsf{froz}_N(P)$ w.r.t. $\equiv$ is finite.*

Proof. By induction on P. □

Not only is $\mathsf{froz}_N(P)$ finite, but, as expressed by the following result, this set is preserved by reduction, in the following sense:

Lemma 4.13. *Let P, Q be two processes such that $P \longrightarrow Q$ or $P \stackrel{\mathsf{cap}}{\to} Q$ for some cap, and assume $\mathrm{fn}(P) \subseteq N$. Then the quotient of $\mathsf{froz}_N(Q)$ w.r.t. $\equiv$ is included in the quotient of $\mathsf{froz}_N(P)$ w.r.t. $\equiv$.*

Proof. We recall that relation $\stackrel{\mathsf{cap}}{\to}$ is defined on the syntax of processes (see Definition 3.1), and the result follows by definition of $\mathsf{froz}_N(P), \mathsf{froz}_N(Q)$.

For $\longrightarrow$, we reason by induction on the derivation of $P \longrightarrow Q$. The cases corresponding to movement transitions follow from $\stackrel{\mathsf{cap}}{\to}$. So the only way a reduction could alter the set of frozen terms is through name substitutions generated by communications, and this is handled by the condition $\mathrm{fn}(P) \subseteq N$. □

4.3. **Local characteristic formulas and completeness.** The purpose of this subsection is to derive local characteristic formulas, defined as follows:

Definition 4.14 (Local characteristic formula)**.** Let $\mathcal{E}$ be a set of terms, P a term and $\mathcal{F}$ a formula. We say that $\mathcal{F}$ is a characteristic formula for P on $\mathcal{E}$ (or, alternatively, a $\mathcal{E}$-characteristic formula for P) if

- $P \models \mathcal{F}$, and
- for any $Q \in \mathcal{E}$, if $Q \models \mathcal{F}$ then $Q \simeq_{\text{int}} P$.

Note that the converse of the second condition always holds, due to soundness of $\simeq_{\text{int}}$ (Theorem 3.29): if $Q \in \mathcal{E}$ and $Q \simeq_{\text{int}} P$, then $Q \models \mathcal{F}$.

With this definition, completeness of $\simeq_{\text{int}}$ boils down to the existence, for any processes P, Q, of a characteristic formula of P on the set $\{Q\}$. Although we do not define directly such a formula, this idea guides the construction of the completeness proof. More precisely, we reason inductively on the sequentiality degree of processes, and manipulate two sets of terms, given a process P:

- $\mathcal{E}_P^{\Downarrow} \stackrel{\text{def}}{=} \{P', \exists\, \mathsf{cap}.\ P \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} P'\}$, that collects the possible evolutions of P,
- and $\mathcal{E}_P^{\text{frz},N} \stackrel{\text{def}}{=} \{P', \mathsf{froz}_N(P') \subseteq \mathsf{froz}_N(P)\}$, that intuitively is the set of processes whose possible evolutions can be captured using the evolutions of P.

We want to establish the existence, for all P, Q, of a local characteristic formula for P on $\mathcal{E}_Q^{\Downarrow}$ and $\mathcal{E}_Q^{\text{frz},N}$. We first prove the following result:

Lemma 4.15. *If a formula $\mathcal{F}$ characterises P on $\mathcal{E}_Q^{frz,N}$ and $N \supseteq \text{fn}(Q)$, then $\mathcal{F}$ characterises P on $\mathcal{E}_Q^{\Downarrow}$.*

Proof. Follows from Lemma 4.13. □

The following lemma describes the construction of a local characteristic formula for guarded terms (of the form $\mathsf{cap}.P$ or $(x)P$) on $\mathcal{E}_Q^{\text{frz},N}$, provided we can compute, given several (smaller) processes R, local characteristic formulas on $\mathcal{E}_R^{\Downarrow}$:

Lemma 4.16. *Consider two processes P and Q, and a set N of names such that $\text{fn}(P) \cup \text{fn}(Q) \subseteq N$. Assume moreover that, for all $Q' \in \mathsf{froz}_N(Q)$, we can construct a formula $\mathcal{F}_{P,Q'}$ characterising P on $\mathcal{E}_{Q'}^{\Downarrow}$ and a formula $\mathcal{F}_{Q',P}$ characterising Q' on $\mathcal{E}_P^{\Downarrow}$. We then have:*

- *for all cap there exists a formula characterising $\mathsf{cap}.P$ on $\mathcal{E}_Q^{frz,N}$,*
- *for all n such that P is not of the form $\{n\} \mid (y)P'$ with $n \notin \text{fn}(P')$, and for all x with $x \notin \text{fv}(P)$, there exists a formula characterising $(x)\big(P\{x/n\}\big)$ on $\mathcal{E}_Q^{frz,N}$.*

Proof.

- Let cap be a given capability. Set $\mathcal{E} = \{Q' \in \mathsf{froz}_N(Q) : \forall P' \text{ s.t. } P \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} P', P' \not\simeq_{\text{int}} Q'\}$; $\mathcal{E} \subseteq \mathsf{froz}_N(Q)$, so by Lemma 4.12, $\mathcal{E}$ is finite, and we can define the formula:
$$F \stackrel{\text{def}}{=} \langle\langle \mathsf{cap} \rangle\rangle \{\!| \bigwedge_{Q' \in \mathsf{froz}_N(Q)} \mathcal{F}_{P,Q'} \,|\!\} \;\wedge\; [\![\mathsf{cap}]\!] \{\!| \bigwedge_{Q' \in \mathcal{E}} \neg \mathcal{F}_{Q',P} \,|\!\}\,.$$
We prove first that $\mathsf{cap}.P \models F$; by hypothesis, $P \models \mathcal{F}_{P,Q'}$ for all $Q' \in \mathsf{froz}_N(Q)$, so that we have $\mathsf{cap}.P \models \langle\langle \mathsf{cap} \rangle\rangle \{\!| \bigwedge_{Q' \in \mathsf{froz}_N(Q)} \mathcal{F}_{P,Q'} \,|\!\}$. Let P' be such that $P \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} P'$, and consider any $Q' \in \mathcal{E}$. Then by hypothesis $P' \models \mathcal{F}_{Q',P}$ would imply $P' \simeq_{\text{int}} Q'$, and hence $Q' \notin \mathcal{E}$, which is contradictory. So $P' \models \bigwedge_{Q' \in \mathcal{E}} \neg \mathcal{F}_{Q',P}$, and finally $P \models F$.

  Conversely, consider $R \in \mathcal{E}_Q^{\text{frz},N}$ such that $R \models F$. We show that $R \simeq_{\text{int}} P$. First, there is Q' such that $R \equiv \mathsf{cap}.Q'$ and $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} \models \mathcal{F}_{P,Q''}$ for all $Q'' \in \mathsf{froz}_N(Q)$. Since $R \in \mathcal{E}_Q^{\text{frz},N}$ and $Q' \in \mathsf{froz}_N(R)$, $Q' \in \mathsf{froz}_N(Q)$, so $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} \models \mathcal{F}_{P,Q'}$, and by hypothesis, $Q' \Longrightarrow \simeq_{\text{int}} P$, which gives the first part of the condition to have $\mathsf{cap}.P \sim_{\text{ind}} R$ (Definition 4.9). Furthermore, since R satisfies the 'necessity' part of the formula F, $Q' \models \bigwedge_{Q'' \in \mathcal{E}} \neg \mathcal{F}_{Q'',P}$, that is $Q' \notin \mathcal{E}$. Thus, there is P' with $P \overset{\langle \mathsf{cap} \rangle}{\Longrightarrow} P'$ and $P' \simeq_{\text{int}} Q'$, which gives the second part of the condition.

- Let n, x be chosen as in the statement of the lemma. We set $P_0 = (x)\big(P\{x/n\}\big)$. Similarly as before, we define $\mathcal{E} = \{Q' \in \mathsf{froz}_N(Q) : \forall P' \text{ s.t. } P \Longrightarrow P',\ P' \not\simeq_{\text{int}} Q'\}$; again $\mathcal{E} \subseteq \mathsf{froz}_N(Q)$, so $\mathcal{E}$ is finite, and we may define the formula:

$$\begin{aligned} F \overset{\text{def}}{=} \ & \neg \copyright n \\ & \wedge \langle\langle ?n \rangle\rangle \big(\mathsf{NonEta} \wedge \textstyle\bigwedge_{Q' \in \mathsf{froz}_N(Q)} \mathcal{F}_{P,Q'}\big) \\ & \wedge [\![?n]\!] \big(\mathsf{NonEta} \longrightarrow \textstyle\bigwedge_{Q' \in \mathcal{E}} \neg \mathcal{F}_{Q',P}\big) \end{aligned}$$

with

$$\mathsf{NonEta} \overset{\text{def}}{=} \neg\big(\{n\} \mid (\neg \copyright n \wedge \langle\langle ?n \rangle\rangle \top)\big)$$

Intuitively, the role of formula NonEta is to detect when the reducts of a process satisfying F stop being eta-equivalent to the initial state.

Let us prove that $P_0 \models F$: $n \notin \text{fn}(P_0)$ by construction, $P_0 \mid \{n\} \Longrightarrow P$, $P \models \mathsf{NonEta}$ and $P \models \bigwedge \mathcal{F}_{P,Q'}$ by hypothesis, so P_0 satisfies the second conjunct in F. Take P' such that $P_0 \mid \{n\} \Longrightarrow P'$ and $P' \models \mathsf{NonEta}$; we prove that $P' \not\models \mathcal{F}_{Q',P}$ for all $Q' \in \mathcal{E}$. Since $P' \models \mathsf{NonEta}$, $P_0 \mid \{n\} \not\equiv P'$, so $P \Longrightarrow P'$. As a consequence, $P' \models \mathcal{F}_{Q',P}$ iff $P' \simeq_{\text{int}} Q'$. Then by definition of $\mathcal{E}$, $P' \models \bigwedge_{Q' \in \mathcal{E}} \neg \mathcal{F}_{Q',P}$. As this holds for all P', we have that $P_0 \models F$.

Let us now prove that if $R \in \mathcal{E}_Q^{\text{frz},N}$ and $R \models F$, then $P \simeq_{\text{int}} R$. Consider such a process R. Then $n \notin \text{fn}(R)$, and there exists Q', R' such that $R \equiv (x)Q'$ and $R \mid \{n\} \Longrightarrow R'$ with $R' \models \mathsf{NonEta} \wedge \bigwedge_{Q' \in \mathsf{froz}_N(Q)} \mathcal{F}_{P,Q'}$. Let $(x)Q''$ be the head eta normal form of $(x)Q'$. By definition, $Q''\{n/x\}$ belongs to $\mathsf{froz}_N(Q)$, and any reduction $(x)Q' \mid \{n\} \Longrightarrow T$ where T is not eta equivalent to $(x)Q' \mid \{n\}$ goes through the state $Q''\{n/x\}$ (i.e., that reduction can be written $(x)Q' \mid \{n\} \Longrightarrow Q''\{n/x\} \Longrightarrow T$). Due to the definition of NonEta, we actually have that $R' \not\equiv_E (x)Q' \mid \{n\}$, so $Q''\{n/x\} \Longrightarrow R'$. Since $R' \models \mathcal{F}_{P,Q''\{n/x\}}$, $R' \simeq_{\text{int}} P$ and the first part of the condition for input in Definition 4.9 is satisfied. Moreover, $R \mid \{n\} \Longrightarrow Q''\{n/x\}$ and $Q''\{n/x\} \models \mathsf{NonEta}$, so $Q''\{n/x\} \models \bigwedge_{Q' \in \mathcal{E}} \neg \mathcal{F}_{Q',P}$. Since $Q''\{n/x\} \in \mathsf{froz}_N(Q)$, we finally have $Q''\{n/x\} \notin \mathcal{E}$, that is there is P' such that $P \Longrightarrow P'$ and $P' \simeq_{\text{int}} Q''\{n/x\}$. This proves the second condition for $P_0 \sim_{\text{ind}} (x)Q''$, and since $(x)Q'' \equiv_E R$, we finally have $P_0 \simeq_{\text{int}} R$. □

We now prove that given P, we can deduce a local characteristic formula for P from local characteristic formulas for its guarded subterms.

Lemma 4.17. *Consider two processes P and Q, and a set of names N, and suppose that, for each subterm of P of the form $\mathsf{cap}.P'$ or $(x)P'$, we can construct a $\mathcal{E}_Q^{frz,N}$-characteristic formula. Then there exists a $\mathcal{E}_Q^{frz,N}$-characteristic formula for P.*

Proof. We assume, without loss of generality, that all occurrences of the replication operator in P are immediately above a guarded process (this is always possible up to $\equiv$).

We construct such a formula $\mathcal{F}_P$ by induction on P. The cases for $\mathbf{0}$, parallel composition, and ambient are easy. Formulas for messages and replicated messages have been given above, and by hypothesis, we have formulas for guarded processes. We are thus left with the case of replicated terms.

If $P = !n[P]$, then $\mathcal{F}_P = !n[\{\!|\ \mathcal{F}'_P\ |\!\}]$ is a $\mathcal{E}_Q^{\text{frz},N}$-characteristic formula, since $\mathcal{F}_{P'}$ is depth selective (all processes satysfying $\mathcal{F}_{P'}$ are intensionally bisimilar to P', so their depth degree is equal to $\mathsf{dd}(P')$ – see Corollary 4.5). If $P = !\mathsf{cap}.\,P'$, then $\mathcal{F}_P = \mathsf{Rep}_{\mathsf{cap}}\{\!|\ \mathcal{F}_{\mathsf{cap}.P'}\ |\!\}$, since $\mathcal{F}_{\mathsf{cap}.P'}$ is sequentially selective. We reason in the same way for the case $P = !(x)P'$. □

Lemma 4.18. *For all P, Q and $N \supseteq \mathrm{fn}(P) \cup \mathrm{fn}(Q)$, there exist characteristic formulas for P on $\mathcal{E}_Q^{\Downarrow}$ and $\mathcal{E}_Q^{frz,N}$.*

Proof. From Lemma 4.15, it is sufficient to construct a local characteristic formula on $\mathcal{E}_Q^{\mathrm{frz},N}$. We remark that without loss of generality, P, Q can be choosed so that every binding $(x)P$ involves a different variable, and this is enough to build characteristic formulas for the set N enriched with distinct names n_x associated to all variables x occurring in P and Q. We reason by induction on $\mathsf{sd}(P)$. If $\mathsf{sd}(P) = 0$, then P has no guarded subterms, and the conditions of Lemma 4.17 are fullfilled, which implies the existence of a local characteristic formula for P.

Assume now $\mathsf{sd}(P) > 0$, and, for all P' such that $\mathsf{sd}(P') < \mathsf{sd}(P)$, and for all Q, there exists a characteristic formula for P' on $\mathcal{E}_Q^{\mathrm{frz},N}$. Consider a process Q. By Lemma 4.17, the existence of a $\mathcal{E}_Q^{\mathrm{frz},N}$-characteristic formula for P can be proved by establishing the existence of a $\mathcal{E}_Q^{\mathrm{frz},N}$-characteristic formula for each guarded subterm of P of the form $\mathsf{cap}.\,P'$ or $(x)P'$. Consider such a guarded subterm $\mathsf{cap}.\,P'$. We have $\mathsf{sd}(P') < \mathsf{sd}(P)$, so by induction there exists a formula $\mathcal{F}_{P,Q'}$ which is a $\mathcal{E}_{Q'}^{\Downarrow}$-characteristic formula for P' for each $Q' \in \mathsf{froz}_N(Q)$. Moreover, by induction, we also have a formula $\mathcal{F}_{Q',P'}$ which is a characteristic formula for Q' on $\mathcal{E}_{P'}^{\Downarrow}$ when $\mathsf{sd}(Q') \leq \mathsf{sd}P') < \mathsf{sd}(P)$. In the case $\mathsf{sd}(Q') > \mathsf{sd}(P')$, we define $\mathcal{F}_{Q',P}$ as the formula $\mathcal{F}_{\mathsf{sd}(Q')}$ given in Lemma 4.3. This formula characterises Q' on $\mathcal{E}_{P'}^{\Downarrow}$: $Q' {\models} \mathcal{F}_{Q',P}$ by Lemma 4.3, and if $P'' \in \mathcal{E}_{P'}^{\Downarrow}$ then $\mathsf{sd}(P'') \leq \mathsf{sd}(P') < \mathsf{sd}(Q')$, so $P'' \not\models \mathcal{F}_{\mathsf{sd}(Q')}$. Hence the requirements of Lemma 4.16 are fullfilled, and there exists a $\mathcal{E}_Q^{\mathrm{frz},N}$-caracteristic formula for $\mathsf{cap}.\,P'$.

Similarly, consider a subterm of the form $(x)P'$, and write $(x)P''$ for its eta normal form. As above, we have local characteristic formulas $\mathcal{F}_{P''\{n_x/x\},Q'}$ and $\mathcal{F}_{Q',P''\{n_x/x\}}$ by induction and using Lemma 4.3 with a similar reasoning. Since $(x)P''$ is in normal form, all requirements of Lemma 4.16 are satisfied, so that there exists a $\mathcal{E}_Q^{\mathrm{frz},N}$-characteristic formula for $(x)P''$, which is also a characteristic formula for $(x)P'$ by Lemma 3.10.

Finally, we have characteristic formulas for all guarded subterms, and by Lemma 4.17, we have a $\mathcal{E}_Q^{\mathrm{frz},N}$-characteristic formula for P. □

Theorem 4.19 (Completeness of $\simeq_{\mathrm{int}}$). *In MA, $=_L \subseteq \simeq_{\mathrm{int}}$.*

Proof. Let P, Q be two terms such that $P \not\simeq_{\mathrm{int}} Q$. By Lemma 4.18, there is a formula F characterising P on $\mathcal{E}_Q^{\Downarrow}$. We have $P \models F$. We then have $Q \in \mathcal{E}_Q^{\Downarrow}$, and $Q \models F$ implies $P \simeq_{\mathrm{int}} Q$. Hence, since by hypothesis $P \not\simeq_{\mathrm{int}} Q$, $Q \not\models F$, and $P \neq_L Q$. □

Corollary 4.20. *In MA, relations $=_L$, $\simeq_{\mathrm{int}}$ and $\sim_{\mathrm{ind}}$ coincide.*

5. Characterizations of logical equivalences

In this section, we compare logical equivalence and standard equivalence relations on processes, like behavioural equivalence and structural congruence. We give an axiomatization of $=_L$ on $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, a subcalculus of MA in which image-finiteness is guaranteed by a syntactical condition (Definition 5.2 below). We shall see that AL is very intensional, in the sense that $=_L$ is 'almost equal' to $\equiv$. More precisely, we show that logical equivalence coincides with $\equiv_E$, the relation obtained by extending structural congruence with the eta law (Definition 3.6). We establish the following chain of (dis)equalities, on $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$:

$$\equiv \subsetneq \equiv_E \; = \; =_L \; = \simeq_{\mathrm{int}} \subsetneq \approx .$$

We then move to the study of a variant of $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ in which communication is synchronous, and show that logical equivalence coincides with $\equiv$ on this calculus. We end this section with a detailed discussion of the treatment of name restricition.

5.1. **Extensionality and intensionality.** We use the characterisation of $=_L$ as $\simeq_{\mathrm{int}}$ to compare logical equivalence with barbed congruence ($\approx$) and structural equivalence ($\equiv$). We start by studying the difference between $=_L$ and $\approx$.

5.1.1. *Non-extensionality.*

Theorem 5.1. *Relation $=_L$ is strictly included in $\approx$.*

Proof. The inclusion follows from $=_L \subseteq \simeq_{\mathrm{int}}$ and $\simeq_{\mathrm{int}} \subseteq \approx$ (the second inclusion is essentially a consequence of the congruence of $\simeq_{\mathrm{int}}$).

The strictness of the inclusion is proved by the following laws, that are valid for $\approx$ but not for $\simeq_{\mathrm{int}}$:
(1) $\mathsf{in}\, n.\, \mathsf{in}\, n = \mathsf{in}\, n \mid \mathsf{in}\, n$
(2) $(x)\,(y)\,\mathbf{0} = (x)\,\mathbf{0} \mid (y)\,\mathbf{0}$
(3) $(x)\,\{x\} = \mathbf{0}$. □

The third axiom is typical for behavioural equivalences in calculi where communication is asynchronous. The first equality can be derived from a more general law, called the *distribution law* in [22]: $M.(P \mid M.P \mid \ldots \mid M.P) = M.P \mid M.P \mid \ldots \mid M.P$ (where M appears the same number of times on both sides of the equality). A similar law is valid for the input prefix, from which the second equality above is derived as an instance. Probably the above are not the only laws that make $=_L$ finer than $\approx$, but a complete axiomatization of $\approx$ over $=_L$ is out of the scope of this paper.

5.1.2. *Intensionality.* We now provide a precise account of the difference between $=_L$ and $\equiv$, in the setting of the subcalculus $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, defined as below. We recall that a process is finite if it does not use the replication operator.

Definition 5.2 ($\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$)**.** The subcalculus $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ is defined by the grammar:

$$P \;::=\; \mathbf{0} \;\big|\; P \mid P \;\big|\; !P \;\big|\; n[P] \;\big|\; \mathsf{cap}.\,P_0 \;\big|\; \{n\} \;\big|\; (x)P_0$$

where P_0 is a finite process.

In $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, we impose finiteness after any form of interaction; in contrast, processes exhibiting an 'infinite spatial structure', such as $!a[b[\mathbf{0}]]$ are allowed.

Lemma 5.3. *All processes of $MA^{\mathrm{s}}_{\mathrm{IF}}$ are image-finite.*

Proof. $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ is included in $\mathrm{MA}_{\mathrm{IF}}$ since the finiteness condition on P_0 in Definition 5.2 implies that $\{P' : P_0 \stackrel{\langle\mathsf{cap}\rangle}{\Longrightarrow} P'\}_{/\simeq_{\mathrm{int}}}$ and $\{P' : P_0\{^n\!/\!_x\} \Longrightarrow P'\}_{/\simeq_{\mathrm{int}}}$ respectively are finite sets. Any process in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ is thus in $\mathrm{MA}_{\mathrm{IF}}$, and is hence image-finite in the sense of Definition 3.26. □

$\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ strictly contains the finite calculus we considered for the completeness proof in Section 3.4.2. Therefore, Theorem 3.33 does not apply, but Corollary 4.20, which holds for the whole calculus, does. As $\mathrm{MA}_{\mathrm{IF}}$, $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ is image-finite, in the sense of Definition 3.26. While in the former subcalculus this property is guaranteed at a semantical level, in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ it follows from a syntactic restriction (we forbid replication in process P_0 – see Definition 5.2).

We will see in Section 6 that $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ is Turing complete.

We let *normalised structural congruence*, written $\equiv_{\mathrm{E}}$, be the relation defined by the rules of $\equiv$ plus the eta law (see Definition 3.6).

Lemma 5.4. $\equiv_E \subseteq \approxeq$.

Proof. It is enough to prove that given P, Q such that $P \longrightarrow_\eta Q$, we have $P \simeq_{\mathrm{int}} Q$. We reason by induction on P, following Lemma 4.8. In that lemma, the situations corresponding to the operators of parallel composition, ambients and capability prefixes are easy because of commutation properties of $\longrightarrow_\eta$. In the cases of $\mathbf{0}$ and of messages, there is no redex for $\longrightarrow_\eta$.

So we only have to examine the clause for the input condition in $\simeq_{\mathrm{int}}$. Let n be a fresh name and write $P \equiv (x)\, P'$, $Q \equiv (x)\, Q'$. We have to prove that $P \mid \{n\} \Longrightarrow \simeq_{\mathrm{int}} Q'\{^n\!/\!_x\}$ and $Q \mid \{n\} \Longrightarrow \simeq_{\mathrm{int}} P'\{^n\!/\!_x\}$. The reduction $P \longrightarrow_\eta Q$ can follow from two reasons: either $P \equiv (x)\,\big(\{x\} \mid (x)\, Q'\big)$, or $P' \longrightarrow_\eta Q'$. In the first case, the proof is straightforward, and in the second case, the induction hypothesis allows us to conclude. □

The converse of this lemma is the difficult part of the characterisation of $=_L$ in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$. This is proved by showing that two intensionally bisimilar finite processes have essentially the same number of prefixes and messages. Using the separative power given by the logic, this entails that $\approxeq \subseteq \equiv_E$ on $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$. It has to be stressed that we rely here on the syntactical finiteness condition defining $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, and that our approach does not apply to, e.g., $\mathrm{MA}_{\mathrm{IF}}$.

We write $\mathtt{messages}(R)$ for the number of messages in R, and $\mathtt{pref}(R)$ for the number of capabilities and abstractions in R.

Lemma 5.5. *Let P, Q be two finite processes. Suppose $P \longrightarrow P'$. Then*
(1) $\mathtt{messages}(P) \geq \mathtt{messages}(P')$;
(2) $\mathtt{pref}(P) \geq \mathtt{pref}(P')$.

Proof. By induction on the derivation of $P \longrightarrow P'$. □

Lemma 5.6. *Let P, Q be two finite processes. Suppose that $P \approxeq Q$, and that both P and Q are eta-normalised. Then* $\mathtt{messages}(P) = \mathtt{messages}(Q)$.

Proof. Suppose $\mathtt{messages}(P) > \mathtt{messages}(Q)$. We prove that we derive a contradiction. We proceed by a case analysis on the shape of P (i.e., the number of its operators)

- $P = P_1 \mid P_2$. Then, by definition of $\approxeq$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approxeq Q_i$. Now, for some i, we should have $\mathtt{messages}(P_i) \neq \mathtt{messages}(Q_i)$, which is impossible, by the induction on the shape.

- $P = \mathsf{cap}.\, P'$. Then, by definition of $\approxeq$, it must be $Q \equiv \mathsf{cap}.\, Q'$ and $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'' \approxeq P'$. It will then be, by Lemma 5.5(1), $\mathtt{messages}(P) = \mathtt{messages}(P') > \mathtt{messages}(Q'')$, which is impossible, by the induction on the shape.
- $P = (x)\, P'$. Then, by definition of $\approxeq$, it must be $Q \equiv (x)\, Q'$; moreover, for n fresh, there must be Q'' such that $\{n\} \mid (x)\, Q' \Longrightarrow Q'' \approxeq P'\{n/x\}$.

  If the reduction $\{n\} \mid (x)\, Q' \Longrightarrow Q''$ contains at least one step, then we would have $\mathtt{messages}(P'\{n/x\}) = \mathtt{messages}(P) > \mathtt{messages}(Q') \geq \mathtt{messages}(Q'')$ and therefore, by induction on the shape, we could not have $Q'' \approxeq P'\{n/x\}$.

  Therefore, suppose $Q'' = \{n\} \mid (x)\, Q'$. Then $Q'' \approxeq P'\{n/x\}$ implies $P'\{n/x\} \equiv \{n\} \mid (x)\, P''$, for some $(x)\, P''$ with n fresh for P and Q. Hence, since n was chosen fresh, the original process P must have been of the form $(x)\, (\{x\} \mid (x)\, P'')$. This means that, modulo $\equiv$, P was not eta-normalised, thus contradicting an hypothesis of the lemma.
- If $P = \{n\}$ then by definition of $\approxeq$ we should have $Q \equiv \{n\}$, which is impossible, since the hypothesis is $\mathtt{messages}(P) > \mathtt{messages}(Q)$. □

Lemma 5.7. *Let P, Q be two finite processes. Suppose $P \approxeq Q$, and that both P and Q are eta-normalised. Then* $\mathtt{pref}(P) = \mathtt{pref}(Q)$.

Proof. Suppose $\mathtt{pref}(P) > \mathtt{pref}(Q)$. We prove that we derive a contradiction. We proceed by induction on the shape of P.

- If $P = \mathbf{0}$ then $Q \equiv \mathbf{0}$.
- $P = P_1 \mid P_2$. Then, by definition of $\approxeq$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approxeq Q_i$. Now, for some i, we should have $\mathtt{pref}(P_i) \neq \mathtt{pref}(Q_i)$, which is impossible, by the induction on the shape.
- $P = \mathsf{cap}.\, P'$. Then, by definition of $\approxeq$, it must be $Q \equiv \mathsf{cap}.\, Q'$ and $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'' \approxeq P'$. Then
$$\mathtt{pref}(P') = \mathtt{pref}(P) - 1 > \mathtt{pref}(Q) - 1 = \mathtt{pref}(Q') \geq \mathtt{pref}(Q'')$$
  Hence $\mathtt{pref}(P') > \mathtt{pref}(Q'')$, which is impossible by the induction on the shape.
- $P = (x)\, P'$. Then, by definition of $\approxeq$, it must be $Q \equiv (x)\, Q'$; moreover, given n fresh, there must be Q'' such that $\{n\} \mid (x)\, Q' \Longrightarrow Q'' \approxeq P'\{n/x\}$.

  Moreover, by the previous lemma we know that $\mathtt{messages}(P) = \mathtt{messages}(Q)$, and we should also have $\mathtt{messages}(P'\{n/x\}) = \mathtt{messages}(Q'')$

  The reduction $\{n\} \mid (x)\, Q' \Longrightarrow Q''$ must contain at least one step, for otherwise we could not have $\mathtt{messages}(P'\{n/x\}) = \mathtt{messages}(Q'')$. For the same reason, during these reductions only the message $\{n\}$ may have been consumed (no other messages). Thus $\{n\} \mid (x)\, Q' \Longrightarrow Q''$ can be written as
$$\{n\} \mid (x)\, Q' \longrightarrow Q'\{n/x\} \Longrightarrow Q''\,,$$
  where $\mathtt{pref}(Q') = \mathtt{pref}(Q'\{n/x\})$ and also $\geq \mathtt{pref}(Q'')$ (Lemma 5.5(2)).

  Therefore we have $\mathtt{pref}(P'\{n/x\}) = \mathtt{pref}(P) - 1 > \mathtt{pref}(Q) - 1 = \mathtt{pref}(Q') \geq \mathtt{pref}(Q'')$. By the induction on the shape, this is in contradiction with $Q'' \approxeq P'\{n/x\}$. □

Lemma 5.8. *Let P, Q be two finite processes. Suppose $P \approxeq Q$, with both P and Q eta-normalised. If $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q' \approxeq P'$. Similarly, if $P \longrightarrow P'$, then there is Q' such that $Q \longrightarrow Q' \approxeq P'$.*

Proof. From Lemmas 5.7 and 5.6: if Q performed more than one action, then it would consume one more prefix or message than P. □

Theorem 5.9. *Let P, Q be processes of MA^{s}_{IF}. Suppose $P \approxeq Q$, with both P and Q eta-normalised. Then $P \equiv Q$.*

Proof. By induction on the shape of P.

- If $P = \mathbf{0}$ then also $Q \equiv \mathbf{0}$.
- Suppose $P = P_1 \mid P_2$. Then, by definition of $\approxeq$, $Q \equiv Q_1 \mid Q_2$ with $P_i \approxeq Q_i$. By induction, $P_i \equiv Q_i$. Hence also $P \equiv Q$.
- Suppose $P = !P'$. Then, by Lemma 4.8, there are r and some $(Q_i)_{1 \leq i \leq r}$ such that
$$Q \equiv !Q_1 \mid (!)Q_2 \mid \ldots \mid (!)Q_r,$$
and $P' \approxeq Q_i$ for all i. By induction, $P' \equiv Q_i$ for all i, so finally $Q \equiv !Q_1 \equiv P$.
- $P = \mathsf{cap}.\, P'$. By definition of $\approxeq$, $Q \equiv \mathsf{cap}.\, Q'$ and there is Q'' such that $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'' \approxeq P'$. By construction of $\mathrm{MA}^{s}_{\mathrm{IF}}$, P', Q' are finite, so that we may apply Lemma 5.8. Then it must be $Q' = Q''$, and therefore by induction $Q' \equiv P'$. We conclude that $P \equiv Q$.
- $P = \{n\}, n[\, P'\,]$: straightforward.
- $P = (x)\, P'$. By definition of $\approxeq$, we have $Q \equiv (x)\, Q'$, and again by construction of $\mathrm{MA}^{s}_{\mathrm{IF}}$, P', Q' are finite. Since $\approxeq$ is a congruence, given n, $\{n\} \mid P \approxeq \{n\} \mid (x)\, Q'$. We have $\{n\} \mid P \longrightarrow P'\{{}^{n}\!/\!_{x}\}$, hence by Lemma 5.8, $\{n\} \mid (x)\, Q' \longrightarrow Q'\{{}^{n}\!/\!_{x}\} \approxeq P'\{{}^{n}\!/\!_{x}\}$. By induction, $P'\{{}^{n}\!/\!_{x}\} \equiv Q'\{{}^{n}\!/\!_{x}\}$; since this holds for any n, $P' \equiv Q'$. □

Corollary 5.10. *Let P, Q be processes of MA^{s}_{IF}. Then $P =_L Q$ iff $P \equiv_{\mathrm{E}} Q$.*

Proof. First, $=_L \subseteq \simeq_{\mathrm{int}}$ by Theorem 3.33, and $\simeq_{\mathrm{int}} \subseteq \equiv_{\mathrm{E}}$ by Theorem 5.9. Conversely, $\equiv_{\mathrm{E}} \subseteq \simeq_{\mathrm{int}}$ by Lemma 5.4, and $\simeq_{\mathrm{int}} \subseteq =_L$ by Theorem 3.29. □

5.2. **Synchronous communications.** We now consider a variant of Mobile Ambients where communication is *synchronous*. For this the production $\{\eta\}$ for messages in the grammar of MA in Table 2.1 is replaced by the production $\{\eta\}.\, P$. Communication is thus synchronous: in $\{\eta\}.\, P$, the process P is blocked until the message $\{\eta\}$ has been consumed. Reduction rule `Red-Com` becomes:

$$\frac{}{\{n\}.\, Q \mid (x)\, P \longrightarrow Q \mid P\{{}^{n}\!/\!_{x}\}}\ \texttt{Red-Com}$$

In the remainder of this subsection, terms belonging to the synchronous version of the calculus will be referred to simply as 'processes'. Since our goal here is to study how the result given by Corollary 5.10 changes when moving to a synchronous calculus, we focus directly on $\mathrm{MA}^{s,s}_{\mathrm{IF}}$, the set of all terms of the synchronous calculus in which processes guarded by prefixes are finite (along the lines of Definition 5.2 that introduces $\mathrm{MA}^{s}_{\mathrm{IF}}$). We shall see that in $\mathrm{MA}^{s,s}_{\mathrm{IF}}$, the eta law fails and the equivalence relation induced by the logic is precisely structural congruence.

In order to show this, we have to port the results about (asynchronous) MA to the synchronous case. The co-inductive characterisation in terms of $\simeq_{\mathrm{int}}$ (that is, Theorems 3.29 and 3.33) remains true, provided that in the definition of intensional bisimulation the communication clauses are replaced by the following:

- If $P \xrightarrow{!n} P'$, then there is Q' such that $Q \stackrel{!n}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$.
- If $P \xrightarrow{?n} P'$ then there is Q' such that $Q \stackrel{?n}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$.

Accordingly, we have to change the definition of syntactical intensional bisimulation by adapting the following clauses for communicating processes:

- If $P \equiv (x)\, P'$ then there is Q' such that $Q \equiv (x)\, Q'$ and for all n there is Q'' such that $Q'\{{}^{n}\!/\!{}_{x}\} \Longrightarrow Q''$ and $P'\{{}^{n}\!/\!{}_{x}\}\mathcal{R}Q''$.
- If $P \equiv \{n\}.\, P'$ then there is Q' such that $Q \equiv \{n\}.\, Q'$ and $Q' \Longrightarrow Q''\mathcal{R}P'$.

As shown in [21], formulas similar to those that are needed in the asynchronous case can be derived for the synchronous calculus. In particular, we have:

Lemma 5.11 ([21]).

- *For all $\mathcal{A}$, there is a formula $\langle\langle ?n\rangle\rangle.\, \{\!|\ \mathcal{A}\ |\!\}$ such that for all P, $P \models \langle\langle ?n\rangle\rangle.\, \{\!|\ \mathcal{A}\ |\!\}$ iff there is P' such that $P \equiv (x)\, P'$ and $P'\{{}^{n}\!/\!{}_{x}\} \Longrightarrow \models \mathcal{A}$.*
- *For all $\mathcal{A}$, there is a formula $\langle\langle !n\rangle\rangle.\, \{\!|\ \mathcal{A}\ |\!\}$ such that for all P, $P \models \langle\langle !n\rangle\rangle.\, \{\!|\ \mathcal{A}\ |\!\}$ iff there is P' such that $P \equiv \{n\}.\, P'$ and $P' \Longrightarrow \models \mathcal{A}$.*

Using this result, the soundness and completeness proofs for $\simeq_{\text{int}}$ with respect to $=_L$ follow exactly the same scheme as in the asynchronous case (see Sections 3 and 4), except that we do not need to reason on eta-normalised terms.

Theorem 5.12 (Soundness and completeness of $\simeq_{\text{int}}$)**.** *Given two processes P and Q of synchronous Mobile Ambients, $P \simeq_{\text{int}} Q$ iff $P =_L Q$.*

We now derive the counterpart of the properties we have established above for $\text{MA}^{\text{s}}_{\text{IF}}$ about the number of messages and prefixes in a term.

Lemma 5.13. *Suppose $P \longrightarrow P'$, where P is a finite process. Then*
(1) $\mathtt{messages}(P) \geq \mathtt{messages}(P')$*;*
(2) $\mathtt{pref}(P) \geq \mathtt{pref}(P')$*.*

Proof. By induction on the derivation of $P \longrightarrow P'$. □

Lemma 5.14. *Let P, Q be two finite processes and suppose $P \approxeq Q$. Then* $\mathtt{messages}(P) = \mathtt{messages}(Q)$*.*

Proof. Suppose $\mathtt{messages}(P) > \mathtt{messages}(Q)$. We prove that we derive a contradiction. We proceed by a case analysis on the shape of P (ie, the number of its operators)

- $P = P_1 \mid P_2$. Then, by definition of $\approxeq$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approxeq Q_i$. Now, for some i, we should have $\mathtt{messages}(P_i) \neq \mathtt{messages}(Q_i)$, which is impossible, by the induction on the shape.
- $P = \mathsf{cap}.\, P'$. Then, by definition of $\approxeq$, it must be $Q \equiv \mathsf{cap}.\, Q'$ and $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Longrightarrow} Q'' \approxeq P'$. It will then be, by Lemma 5.5(1), $\mathtt{messages}(P) = \mathtt{messages}(P') > \mathtt{messages}(Q'')$, which is impossible, by the induction on the shape.
- $P = \{n\}.\, P'$. Then $Q \equiv \{n\}.\, Q'$ and $P' \approxeq Q'$. But $\mathtt{messages}(P') > \mathtt{messages}(Q')$, which by induction is impossible.
- $P = (x)\, P'$. Then $Q \equiv (x)\, Q'$ and for all h fresh, $Q'\{{}^{h}\!/\!{}_{x}\} \approxeq\!\Longrightarrow Q''$ and $P'\{{}^{h}\!/\!{}_{x}\} \approxeq Q''$ and $\mathtt{messages}(P'') > \mathtt{messages}(Q'')$, so we can conclude by induction. □

Lemma 5.15. *Let P, Q be two finite processes, and suppose $P \approxeq Q$. Then* $\mathtt{pref}(P) = \mathtt{pref}(Q)$*.*

Proof. Suppose $\mathtt{pref}(P) > \mathtt{pref}(Q)$. We prove that we derive a contradiction. We proceed by induction on the shape of P.

- If $P = \mathbf{0}$ then $Q \equiv \mathbf{0}$.

- $P = P_1 \mid P_2$. Then, by definition of $\approxeq$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approxeq Q_i$. Now, for some i, it should be $\mathtt{pref}(P_i) \neq \mathtt{pref}(Q_i)$, which is impossible, by the induction on the shape.
- $P = \mathsf{cap}.\, P'$. Then, by definition of $\approxeq$, it must be $Q \equiv \mathsf{cap}.\, Q'$ and $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'' \approxeq P'$. Then
$$\mathtt{pref}(P') = \mathtt{pref}(P) - 1 > \mathtt{pref}(Q) - 1 = \mathtt{pref}(Q') \geq \mathtt{pref}(Q'')$$
Hence $\mathtt{pref}(P') > \mathtt{pref}(Q'')$, which is impossible by the induction on the shape.
- $P = \{n\}.\, P'$. Similar to capability case.
- $P = (x)\, P'$. Then $Q \equiv (x)\, Q'$ and there is Q'' such that $Q'\{^h\!/\!_x\} \approxeq\!\Longrightarrow Q''$ and $P'\{^h\!/\!_x\} \approxeq Q''$. There is no consumption of messages, hence $\mathtt{pref}(P'\{^h\!/\!_x\}) > \mathtt{pref}(Q'')$, and we can conclude using induction. □

Lemma 5.16. *Let P, Q be two finite processes, and suppose $P \approxeq Q$. If $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q' \approxeq P'$. Similarly, if $P \longrightarrow P'$, then there is Q' such that $Q \longrightarrow Q' \approxeq P'$.*

Proof. From the two previous lemmas: if Q performed more than one action, then it would consume one more prefix or message than P. □

Theorem 5.17. *Let P, Q be two processes in $MA^{\mathrm{s,s}}_{\mathrm{IF}}$, and suppose $P \approxeq Q$. Then $P \equiv Q$.*

Proof. By induction on the shape of P (almost exactly as in Theorem 5.9). □

Corollary 5.18. *Let P, Q be processes of $MA^{\mathrm{s,s}}_{\mathrm{IF}}$. Then $P {=_L} Q$ iff $P \equiv Q$.*

5.3. **Name restriction.** In this section, we consider the variant of MA, noted here MA^{ν}, that includes name restriction $(\nu n)\, P$. We discuss, among previous results, which ones remain valid, and which ones have to be amended.

Adding name restriction involves several modifications in the definition of the calculus and of the logic. Name n is bound in $(\nu n)\, P$, and the definition of $\mathrm{fn}(P)$ is modified accordingly. Regarding structural congruence, we add alpha conversion for ν, as well as the following laws:

$$(\nu n)\, \mathbf{0} \equiv \mathbf{0} \qquad (\nu n)(\nu m)\, P \equiv (\nu m)(\nu n)\, P \qquad (\nu n)\, (P \mid Q) \equiv P \mid (\nu n)\, Q \text{ if } n \notin \mathrm{fn}(P)$$
$$(\nu n)\, m[P] \equiv m[(\nu n)\, P] \qquad \mathsf{cap}.\, (\nu n)\, P \equiv (\nu n)\, \mathsf{cap}.\, P \text{ if } n \notin \mathrm{fn}(\mathsf{cap})$$

The last rule is not always present in the definition of structural congruence. It is not an essential rule, but including it makes our some technical details simpler.

In the logic, additional connectives are introduced, as in [12], to handle restriction and the associated notion of freshness of names: formulas can also be of the form $n \circledR \mathcal{A}$, $\mathcal{A} \oslash n$, or $\text{И} n.\, \mathcal{A}$. Accordingly, the enriched notion of satisfaction, written $\models^{\nu}$, is given by:

- $P \models^{\nu} n \circledR \mathcal{A}$ iff $P \equiv (\nu n)\, P'$ and $P' \models^{\nu} \mathcal{A}$ for some P';
- $P \models^{\nu} \mathcal{A} \oslash n$ if $(\nu n)\, P \models^{\nu} \mathcal{A}$;
- $P \models^{\nu} \text{И} n.\, \mathcal{A}$ if there is $n' \notin (\mathrm{fn}(P) \cup \mathrm{fn}(\mathcal{A}))$ such that $P \models^{\nu} \mathcal{A}\{^{n'}\!/\!_n\}$.

To illustrate this new setting, we consider the two following formulas:

$$\mathsf{free}(n) \stackrel{\text{def}}{=} \neg n \circledR \top \qquad\qquad \mathsf{public} \stackrel{\text{def}}{=} \text{И} n.\, \neg\big(n \circledR\, \mathsf{free}(n)\big) \ .$$

A process P satisfying $\mathsf{free}(n)$ cannot reveal n, which means that n necessarily occurs free in P. In turn, if P satisfies public, then it cannot reveal a name n so as to exhibit free occurrences of n, which means that P is structurally congruent to some $P' \in \mathrm{MA}$.

Formula public hence provides a way of selecting processes belonging to MA among the processes in MA^ν. We can indeed adapt any formula $\mathcal{A}$ we have used in the paper into a formula $\mathcal{A}'$ such that whenever $P \models^\nu \mathcal{A}'$, then $P \equiv P'$ for some P' in MA such that $P' \models \mathcal{A}$; in particular, formulas of the form $\mathcal{A}_1 \triangleright \mathcal{A}_2$ are translated into formulas of the form $(\mathcal{B}_1 \wedge \mathsf{public}) \triangleright \mathcal{B}_2$.

In presence of name restriction, we can adapt rather easily several important results of the paper as follows (for each item, we indicate the part of the paper we refer to):

- a new 'intensional' rule must be added to the definition of $\simeq_{\mathrm{int}}$ (Def. 3.2): if $P \equiv (\nu n)\, P'$, then there is Q' such that $Q \equiv (\nu n)\, Q'$ and $P' \simeq_{\mathrm{int}} Q'$;
- with this definition, it is possible to establish a soundness result ($\simeq_{\mathrm{int}} \subseteq \; =_L$, Theorem 3.29), and completeness for finite processes (processes without replication, Theorem 3.33);
- characteristic formulas are derivable for processes of the form $(\nu n_1) \ldots (\nu n_k)\, P$, where P is a 'public' process in $\mathrm{MA}_{\mathrm{IF}}$ (Lemma 3.27): we rely on name revelation to get rid of the topmost restrictions, and then translate the characteristic formula for P using the approach sketched above;
- logical equivalence coincides with structural congruence enriched with eta conversion for processes of the form $(\nu n_1) \ldots (\nu n_k)\, P$, with P a public process in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ (Corollary 5.10).

The difficult point, that we leave for future work, is to analyse processes that can generate unboundedly many names, i.e., in which restriction occurs under replication. Characteristic formulas seem much more difficult to obtain for such processes. We do not know at present how to derive completeness in absence of an image finiteness hypothesis (in particular, we do not see how a counterpart of Lemma 4.13 can be obtained).

6. (Un)decidability of logical equivalence

In this section we define the encoding of a Turing Machine in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$. The purpose of this encoding is to establish that logical equivalence in undecidable on $\mathrm{MA}_{\mathrm{IF}}$.

The definition of the encoding requires the introduction of some constructions that will be given as ($\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$) contexts. To ease the reading of our definitions, we shall sometimes work with *parametrised contexts*, which are context definitions that depend on some values (names, words, or movements of the head of the Turing Machine). Additionally, some parametrised definitions shall be written $\mathtt{foo}(p); P$: here, `foo` is the name of the definition, whereas p and P are parameters (P being a process); the notation emphasizes the sequentiality between the process being introduced and P.

Remark 6.1. The results in this section improve and extend a preliminary version presented in [20]. By the time the writing of this paper was completed, Busi and Zavattaro [3] have studied encodings of another universal machine, namely the Random Access Machine, into a subset of MA. Their encodings are syntactically more coincise than the one below of a Turing Machine. However, Busi and Zavattaro make use of combinations of operators that are not licit in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ (i.e., their encodings are not encodings into $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$). Also, while longer, the encoding of Turing Machines makes use of components which accomplish simple tasks and which interact with each other in simple manners. Correspondingly, each step of the proof, which follows the reductions of the encoding of a Turing Machine, is rather straightforward. For these reasons we maintain the schema of the original encoding in [20].

6.1. **Ribbons. Digits and words.** We associate to booleans true and false two names tt and $f\!f$. We call these names *digits*, and range over digits with d, d'. A word will be the result of a (possibly empty) concatenation of digits. The empty word shall be written ϵ. We range over words with w, w', w_1, w_2. Given a word w consisting in r digits (with $r \geq 1$), we shall sometimes write $w^1 \dots w^r$ to refer to the digits of w. This should not be confused with notation $f\!f^n$, that we will sometimes use to represent the word consisting in n times digit $f\!f$ (this should be clear from the context).

We start with the definition of the support of the Turing Machine: ribbons can be in differents states (frozen, growing, work ribbon, old), and are defined as follows:

Cells and Words

$$\begin{array}{lcl}
\mathtt{cell}(d)\{\!\!|\ |\!\!\} & := & cell[\,d[\,\mathbf{0}\,] \mid !\mathsf{open}\ wo \mid \{\!\!|\ |\!\!\}\,] \\
\mathtt{word}(w)\{\!\!|\ |\!\!\} & := & \mathtt{cell}(w^1)\{\!\!|\ \mathtt{cell}(w^2)\{\!\!|\ \dots \mathtt{cell}(w^r)\{\!\!|\ |\!\!\}\dots |\!\!\}\ |\!\!\} \qquad (w = w^1 w^2 \dots w^r)
\end{array}$$

Ribbon Extensor

$$\begin{array}{lcl}
\mathtt{deadextcode} & := & !\mathsf{open}\ coin.\,\mathsf{open}\ newcell.\,\mathsf{in}\ cell.\,coin[\,\mathbf{0}\,] \\
 & & \mid\ !newcell[\,\mathtt{cell}(f\!f)\{\!\!|\ \mathsf{out}\ ext\ |\!\!\}\,] \\
\mathtt{sendstart} & := & msg[\,\mathsf{out}\ ext.\,!\mathsf{out}\ cell \mid \mathsf{out}\ ribbon_left.\,start[\,\mathsf{in}\ TM\,]\,] \\
\mathtt{ExtensorFrozen} & := & ext[\,\mathtt{deadextcode} \mid \mathsf{open}\ coin.\,\mathtt{sendstart}\,] \\
\mathtt{ExtensorAlive} & := & ext[\,coin[\,\mathbf{0}\,] \mid \mathtt{deadextcode} \mid \mathsf{open}\ coin.\,\mathtt{sendstart}\,] \\
\mathtt{ExtensorDead} & := & ext[\,\mathtt{deadextcode}\,]
\end{array}$$

Ribbons

$$\begin{array}{lcl}
\mathtt{cleaninst} & := & \mathsf{open}\ cleaner.\,\mathsf{open}\ runclean \mid runclean[\,\mathtt{deadcleancode}\,] \\
\mathtt{deadcleancode} & := & !\mathsf{open}\ f\!f \mid !\mathsf{open}\ tt \mid !\mathsf{open}\ cell \mid !\mathsf{open}\ wo \\
\mathtt{FrozenRibb}(w) & := & ribbon_left[\,\mathtt{cleaninst} \mid \mathtt{word}(w)\{\!\!|\ \mathtt{ExtensorFrozen}\ |\!\!\}\,] \\
\mathtt{GrowingRibb}(w) & := & ribbon_left[\,\mathtt{cleaninst} \mid \mathtt{word}(w)\{\!\!|\ \mathtt{ExtensorAlive}\ |\!\!\}\,] \\
\mathtt{WorkRibb}(w_1, w_2)\{\!\!|\ |\!\!\} & := & ribbon_left[\,\mathtt{cleaninst}\,] \\
 & & \mid \mathtt{word}(w_1)\{\!\!|\ \{\!\!|\ |\!\!\} \mid \mathtt{word}(w_2)\{\!\!|\ \mathtt{ExtensorDead}\ |\!\!\}\ |\!\!\} \\
\mathtt{OldRibb} & := & ribbon_left[\,\mathtt{deadcleancode} \mid \mathtt{ExtensorDead}\,]
\end{array}$$

All names used in the definitions above are supposed to be pairwise distinct. In particular, TM is the name we shall use for the ambient containing the Turing Machine (see Definition 6.5). The ribbon is represented as a nesting of ambients named *cell*, each of which contains an empty ambient named d, where d is the digit value of the cell: this corresponds to the definitions of $\mathtt{cell}(d)$ and `word` – the $!\mathsf{open}\ wo$ subterm is there to trigger the computation of the head of the machine as soon as the head 'points to' (i.e., enters) the current cell (see Section 6.2).

Ribbon extension is used to generate a sufficiently long nesting of *cell* ambients for the machine to run. A frozen ribbon consists of a word w, containing at the end of the ribbon a frozen ribbon extensor (definition of `FrozenRibb` – the `cleaninst` part will be useful later on). The extensor is triggered by the presence of an ambient named *coin* (definitions `ExtensorFrozen` and `ExtensorAlive`): when this happens, the loop programmed in the definition of `deadextcode` can start, which can have the effect of adding new cells, whose value is ff. Each time the extensor loops (state `ExtensorAlive`), the *coin* ambient can be erased by process open *coin*. `sendstart`, which has the effect of stopping the extension process, and sending an ambient *msg* out of the ribbon to instruct the machine to start computation. When this happens, the extensor is in `ExtensorDead` state.

A ribbon in `GrowingRibb` state keeps extending until the extensor dies, at which point it becomes a `WorkRibb` (`WorkRibb` has two parameters, w_1 and w_2, in order to reason about the cell where the head of the machine currently is). Along this evolution, the `cleaninst` code is always present. When the machine successfully terminates computation (we will describe below how this happens), it generates an ambient named *cleaner*, which triggers the cleaning of the machine: all ambients $cell, tt, \mathit{ff}, wo$, that intuitively constitute the "data structures" of the machine, are removed. At this point, we obtain an `OldRibb`.

Some of the explanations we have just given are formalised by the following result, which will be used to establish undecidability of $=_L$.

Lemma 6.2 (Ribbon evolution).
For any word w and $n \in \mathbb{N}$, we write $P_n = \mathtt{GrowingRibb}(w.(\mathit{ff})^n)$, where $(\mathit{ff})^n$ stands for the word written as n times the name ff. We have:

- $P_n \Longrightarrow P_{n+1}$;
- $P_n \Longrightarrow R$ *with* $R = \mathtt{WorkRibb}(\epsilon, w.(\mathit{ff})^n)\{\!|\ msg[\,!\mathsf{out}\ cell \mid \mathsf{out}\ ribbon\_left.\, start[\,\mathsf{in}\ TM\,]\,]\ |\!\}$;
- *for any term Q along the reduction paths from P_n to P_{n+1} and from P_n to R, there exists Q' such that $Q \equiv ribbon\_left[\,Q'\,]$.*

Moreover, for any word w, we have:

$$\mathtt{WorkRibb}(w, \epsilon)\{\!|\ \mathbf{0}\ |\!\} \mid cleaner[\,\mathsf{in}\ ribbon\_left\,] \Longrightarrow \mathtt{OldRibb}.$$

Proof. At any step, the extensor can only choose between creating a new ff cell or dying and sending up through the ribbon an ambient *msg*. Note that when extending the ribbon with a new ff cell, there are at some point two concurrent actions in *cell* and out *ext*: these are in causal dependency, since the in *cell* can only happen once the out *ext* has taken place, which ensures sequentiality of the execution. □

6.2. **Turing Machine.**

Definition 6.3 ((Ideal) Turing Machine). We introduce three symbols $\leftarrow, \downarrow$ and $\rightarrow$ for the movements of the head of a Turing Machine.

We represent a Turing Machine as a quadruplet $(\mathcal{Q}, q_{start}, q_A, \delta)$ where $\mathcal{Q}$ is a set of states, q_{start} is the initial state, q_A is the accepting state, and $\delta : \mathcal{Q} \times \{\mathit{ff}, tt\} \longrightarrow \mathcal{Q} \times \{\mathit{ff}, tt\} \times \{\leftarrow, \downarrow, \rightarrow\}$ is the evolution function.

Notation: we shall write

$$(w_1, q, w_2) \rightarrowtail\!\!\!\rightarrowtail (w_1', q', w_2')$$

to denote the fact that the Turing Machine in state q with the head on the cell of the last letter of w_1 (which will be referred to as *"the head dividing the ribbon into words w_1 and w_2"*) evolves in one step of computation into the machine in state q', dividing the ribbon into words w_1' and w_2'.

The remainder of this subsection is devoted to establishing the following claim:

Turing Machine Transitions

$$\begin{array}{lcl}
\mathtt{clear}(d);P & := & wo[\ \mathsf{out}\ head.\,\mathsf{open}\ d.\,cl_ack[\,\mathsf{in}\ head\,]\ ]\ \mid\ \mathsf{open}\ cl_ack.\,P\\
\mathtt{write}(d);P & := & wo[\ \mathsf{out}\ head.\,d[\,\mathbf{0}\,]\ \mid\ wr_ack[\,\mathsf{in}\ head\,]\ ]\ \mid\ \mathsf{open}\ wr_ack.\,P\\
\mathsf{become}(mo);P & := & mo[\ \mathsf{out}\ head.\,\mathsf{open}\ head.\,P\ ]\ \mid\ \mathsf{in}\ mo\\
\mathsf{domove}(mv);P & := & \begin{cases} \mathsf{in}\ cell.\,P & \text{if } mv = \leftarrow\\ P & \text{if } mv = \downarrow\\ \mathsf{out}\ cell.\,P & \text{if } mv = \rightarrow \end{cases}\\
\mathtt{tcode}(d_r, q_w, d_w, mv) & := & \mathtt{clear}(d_r);\mathtt{write}(d_w);\\
 & & \qquad \mathsf{become}(mo);\mathsf{in}\ TM.\,\mathsf{domove}(mv);\mathsf{open}\ q_w
\end{array}$$

State

$$\begin{array}{lcl}
f\!f \longrightarrow P\ +\ tt \longrightarrow Q & := & coin[\,\mathsf{in}\ f\!f.\,\mathsf{out}\ f\!f.\,P\,]\ \mid\ coin[\,\mathsf{in}\ tt.\,\mathsf{out}\ tt.\,Q\,]\ \mid\ \mathsf{open}\ coin\\
\mathsf{code}(q) & := & !q[head[\ \mathsf{out}\ TM.\,\big(\ f\!f \longrightarrow \mathtt{tcode}(f\!f, d_{f\!f}, q_{f\!f}, mv_{f\!f})\\
 & & \qquad\qquad\qquad +\ tt \longrightarrow \mathtt{tcode}(tt, d_{tt}, q_{tt}, mv_{tt})\big)\ ]]\\
 & & \mid\ !coin[\,\mathsf{in}\ f\!f.\,\mathsf{out}\ f\!f.\,\mathtt{tcode}(f\!f, d_{f\!f}, q_{f\!f}, mv_{f\!f})\,]\\
 & & \mid\ !coin[\,\mathsf{in}\ tt.\,\mathsf{out}\ tt.\,\mathtt{tcode}(tt, d_{tt}, q_{tt}, mv_{tt})\,]\\
\mathsf{code}(q_A) & := & !q_A[\ get_out[\,\mathbf{0}\,]\ ]
\end{array}$$

Turing Machine Behavior after Recognition

$$\begin{array}{l}
\mathtt{getout}\ :=\\
\quad !\mathsf{open}\ get_out.\,\mathsf{out}\ cell.\,get_out[\,\mathbf{0}\,]\\
\mid\ !\mathsf{open}\ get_out.\,\mathsf{out}\ ribbon_left.\,\big(\ cleaner[\,\mathsf{out}\ TM.\,\mathsf{in}\ ribbon_left\,]\\
\qquad\qquad\qquad \mid\ coin[\,\mathsf{out}\ TM.\,\mathsf{in}\ ribbon_left.\,\mathsf{in}\ cell^{\mathsf{length}(w)}.\,\mathsf{in}\ ext\,]\\
\qquad\qquad\qquad \mid\ \mathsf{open}\ start.\,\mathsf{in}\ ribbon_left.\,\mathsf{in}\ cell.\,\mathsf{open}\ q_{start}\ \big)
\end{array}$$

Figure 1: Encoding Turing Machines in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$

Claim 6.4. *Any Turing Machine computation may be encoded in $MA^{\mathrm{s}}_{\mathrm{IF}}$.*

To encode Turing Machines, we must describe how we simulate in $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ the transitions of the machine, and how some extra manipulations are performed after recognition of a word (these are necessary to deduce the undecidability result proved below).

The encoding is given by the definitions collected in Figure 1. The overall shape of the encoding can be described as follows:

Definition 6.5 (Turing Machine in Mobile Ambients)**.** The encoding of a Turing Machine is based on an ambient named TM, containing a persistent process named `tmsoup`:

$$\mathtt{tmsoup}\ :=\ \mathsf{code}(q_0) \mid \ldots \mid \mathsf{code}(q_n)\ \mid\ \mathtt{getout}\ \mid\ !\mathsf{open}\ mo\,.$$

We define two configurations for the encoding of a Turing Machine. Before being active, the machine is in *starting state*, defined by:

$$\mathtt{TMStart}\ :=\ TM[\ \mathsf{open}\ start.\,\mathsf{in}\ ribbon\_left.\,\mathsf{in}\ cell.\,\mathsf{open}\ q_{start}\ \mid\ \mathtt{tmsoup}\ ]\,.$$

Once the computation has started, the Turing Machine in state q is represented by the term

$$\mathtt{TM}(q)\ :=\ TM[\ \mathsf{open}\ q\ \mid\ \mathtt{tmsoup}\ ]\,.$$

Lemma 6.6 ($\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ encoding)**.** *All terms used in the encoding of a Turing Machine belong to $MA^{\mathrm{s}}_{\mathrm{IF}}$.*

Our Turing Machine encoding is somehow reminiscent of the one presented in [13]. We should however remark that we work here in a language without name restriction, and with a simpler encoding of choice (operator + above, to test the value of a cell).

According to the explanations given in Section 6.1, the machine reacts to the presence of an ambient named *start* to enter the first cell of the ribbon and start computation (definition TMStart).

The behaviour of the running machine is described by the definition of $\mathsf{code}(q)$: the head of the machine enters the current cell, and tests its value by concurrently trying to enter ambients named *ff* and *tt*. According to the ambient being present, the appropriate machine transition is triggered (definition of tcode — d_{ff}, q_{ff}, mv_{ff} stand for the new value, new state, and movement of the head determined by the current state if the value read is *ff*, and similarly for *tt*). The last two lines in the definition of code (processes starting with $!coin \dots$) are there for garbage collection purposes: they "absorb" the branch of the choice that has not been triggered.

Performing a transition involves erasing the current value of the cell, installing the new value, getting back inside the Turing Machine (the current working ambient had to get out of it to read the value of the cell), and triggering the movement of the machine (definition of tcode). The corresponding definitions on top of Figure 1 should be self-explanatory, the $\mathsf{become}(mo)$ part being necessary to synchronise with the $!\mathsf{open}\ mo$ inside ambient TM. Finally, $\mathsf{open}\ q_w$ starts the execution of the code corresponding to q_w, the new state of the machine — according to Definition 6.5, the code of all possible states of the machine is present in replicated form in TM.

The code of the accepting state q_A is peculiar: when the machine reaches this state, it triggers process getout, which makes it exit the ribbon and start the cleaning process. As explained above, the presence of an ambient named *cleaner* in ambient *ribbon_left* triggers process cleaninst of Section 6.1. The process on the last line of Figure 1 is there to install the machine in the *exact* initial state once the word has been recognized and cleaning has been performed. This is necessary to obtain a loop in the proof of Lemma 6.13 below.

We can remark that the encoding is parametric over a word w, whose length (denoted $\mathsf{length}(w)$) is used in the definition of getout (in that definition, $\mathsf{in}\ cell^{\mathsf{length}(w)}$ stands for the concatenation of $\mathsf{length}(w)$ copies of the capability $\mathsf{in}\ cell$). This aspect of our encoding is however irrelevant since it is influent only after the end of the execution of the machine, and not during the central part of the simulation.

We now formulate the evolution of the terms we have defined in order to simulate Turing Machines. We first introduce a useful relation.

Definition 6.7 (deterministic evolution relation)**.** We say that a process P *deterministically evolves* to Q, written $P \rightsquigarrow Q$, if and only if $P \longrightarrow Q$ and for any Q' s.t. $P \longrightarrow Q'$, either $Q' \not\longrightarrow$ or $Q \equiv Q'$.

Notation: We shall write $P \rightsquigarrow^k Q$ to say that P deterministically reduces to Q in k steps ($k \geq 1$). We write $P \rightsquigarrow^+ Q$ when $P \rightsquigarrow^k Q$ for some k.

Using $\rightsquigarrow$, we can state some elementary facts about the macros involved in the execution of the machine. The relation $P \rightsquigarrow^+ Q$ captures the fact that P cannot avoid reducing to Q except for some immediately blocking states. Such blocking states may only appear due to the firing of the "wrong branch" in a choice encoding ($ff \longrightarrow \cdots + tt \longrightarrow \dots$). (Incidentally, we may remark that a purely deterministic encoding of the Turing Machine could probably be definable, but at the cost of more complex definitions and proofs.)

Lemma 6.8 (state evolution). *For any terms P, Q, names $d, d' \in \{ff, tt\}$ and word w, we set $M = d[\,\mathbf{0}\,] \mid !\mathsf{open}\ wo \mid \mathtt{word}(w)\{\!|\ \mathtt{ExtensorDead}\ |\!\}$. We then have the following deterministic transition sequences:*

(1) $head[\,d \longrightarrow P + \neg d \longrightarrow Q\,] \mid d[\,\mathbf{0}\,] \mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$

$\rightsquigarrow^3\ head[\,P \mid coin[\,\mathsf{in}\ \neg d.\,\mathsf{out}\ \neg d.\,Q\,]\,] \mid d[\,\mathbf{0}\,] \mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$;

(2) $head[\,\mathtt{clear}(d); P \mid coin[\,\mathsf{in}\ d'.\,Q\,]\,] \mid d[\,\mathbf{0}\,] \mid\mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$

$\rightsquigarrow^5\ head[\,P \mid coin[\,\mathsf{in}\ d'.\,Q\,]\,] \mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$;

(3) $head[\,\mathtt{write}(d); P \mid coin[\,\mathsf{in}\ d'.\,Q\,]\,] \mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$

$\rightsquigarrow^4\ head[\,P \mid coin[\,\mathsf{in}\ d'.\,Q\,]\,] \mid d[\,\mathbf{0}\,] \mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$;

(4) $head[\,\mathtt{become}(mo); P \mid coin[\,\mathsf{in}\ d'.\,Q\,]\,] \mid d[\,\mathbf{0}\,] \mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$

$\rightsquigarrow^3\ mo[\,P \mid coin[\,\mathsf{in}\ d'.\,Q\,]\,] \mid d[\,\mathbf{0}\,] \mid !\mathsf{open}\ wo \mid cell[\,M\,] \mid TM[\,\mathtt{tmsoup}\,]$.

Moreover, the same results hold with a frozen (instead of dead) extensor in M, the only condition being that ambient ext contains an inactive term.

Proof. By inspection of the possible reductions of the processes being considered. From the second statement on, the ambient $coin[\,\mathsf{in}\ d'.\,Q\,]$ is frozen: it actually represents the non-chosen branch in the encoding of the choice operator, that will be erased later, when the head of the Turing Machine comes back inside ambient TM (see below). □

We can now merge the results above into a property regarding transitions of the Turing Machine.

Lemma 6.9 (One step of Turing Machine simulation).

Let $\mathcal{M}$ be a Turing Machine, q one of its non accepting states, and w_1, w_2 two words, with $w_2 \neq \epsilon$. Suppose $(w_1, q, w_2) \rightarrowtail\!\!\!\!\rightarrowtail (w_1', q', w_2')$. Then

$$\mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TM}(q)\ |\!\} \rightsquigarrow^+ \mathtt{WorkRibb}(w_1', w_2')\{\!|\ \mathtt{TM}(q')\ |\!\}\,.$$

Proof. We divide the evolution of the term representing the Turing Machine into the following steps:

(1) From state q, the TM can trigger the q code by performing the corresponding open operation, which has the effect of releasing an ambient named *head*. Moreover, this is the only place where some reduction is possible, because first, $\mathtt{Extensor}$ is inactive and second, in every ambient named *cell*, no reduction occurs. Therefore,

$$\begin{array}{l}\mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TM}(q)\ |\!\} \\ \quad \rightsquigarrow^2\ \mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TMNostate} \mid head[\,ff \longrightarrow \cdots + tt \longrightarrow \dots\,]\ |\!\}\end{array}$$

where the notation $\mathtt{TMNostate}$ stands for the following configuration of the Turing Machine ambient:

$$TM[\ \mathtt{code}(q_0) \mid \dots \mid \mathtt{code}(q_n) \mid \mathtt{tmsoup}\ ]$$

Note that this ambient cannot perform any reduction as long as it is not visited by a *mo* or *getout* ambient.

(2) Using the previous fact, and considering that reductions can only take place at *cell* level, we have

$$\begin{aligned}&\mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TMNostate}\ \mid\ head[\,\mathit{ff}\longrightarrow\cdots+tt\longrightarrow\dots]\ |\!\}\ \leadsto^{15}\\ &\mathtt{WorkRibb}(w_1^1\dots w_1^{r-1}d, w_2)\{\!|\ \mathtt{TMNostate}\\ &\qquad\qquad\qquad \mid mo[\,\mathsf{in}\ TM.\,\mathtt{domove}(mv).\,\mathsf{open}\ q'\ \mid\ coin[\,\mathsf{in}\ \neg w_1^r.\,P\,]\,]\ |\!\}\end{aligned}$$

where $\delta(q, w_1^r) = (q', d, mv)$ (i.e., the machine evolves from q to q' when reading w_1^r).

(3) The ambient mo comes back into the Turing Machine and is opened by the $\mathtt{tmsoup}$ component. Then the head movement (if any) is performed, which activates an $\mathsf{open}\ q'$ process, so that the Turing Machine gets into $\mathtt{TM}(q')$ state.

$$\begin{aligned}&\mathtt{WorkRibb}(w_1^1\dots w_1^{r-1}d, w_2)\{\!|\ \mathtt{TMNostate}\\ &\qquad \mid mo[\,\mathsf{in}\ TM.\,\mathtt{domove}(mv).\,\mathsf{open}\ q'\ \mid\ coin[\,\mathsf{in}\ \neg w_1^r.\,P\,]\,]\ |\!\}\\ &\qquad\qquad\qquad \leadsto^{2(+1)}\ \mathtt{WorkRibb}(w_1', w_2')\{\!|\ \mathtt{TM}(q')\ |\!\}\,.\end{aligned}$$

Note that opening ambient mo triggers the absorbtion of the non-selected branch of the choice (ambient $coin$) by a $!coin[\dots]$ (from the code for the original state of the machine).

The 2(+1) above comes from the fact that the head of the machine can also make no movement in its transition from a state to another (case $\downarrow$). ☐

We obtain as a corollary of the Lemma above:

Proposition 6.10 (Turing Machine simulation). *Given a Turing Machine $\mathcal{M}$, for any word w and $n \in \mathbb{N}$, the Turing Machine $\mathcal{M}$ recognises the word w on the ribbon $w.\,\mathit{ff}^n$ iff there exist two words w_1 and w_2 s.t.*

$$\mathtt{WorkRibb}(\epsilon, w.\,\mathit{ff}^n)\{\!|\ \mathtt{TM}(q_{start})\ |\!\}\ \leadsto^{+}\ \mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TM}(q_A)\ |\!\}\,,$$

where the terms above are given by the encoding of $\mathcal{M}$.

Let us finally describe what happens after the machine has reached the accepting state.

Lemma 6.11 (Acceptation). *Let w_1, w_2 be two words. Then*

$$\begin{aligned}&\mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TM}(q_A)\ |\!\}\\ &\qquad \Longrightarrow \mathtt{OldRibb}\ \mid\ \mathtt{TMStart}\ \mid\ coin[\,\mathsf{in}\ ribbon\_left.\,\mathsf{in}\ cell^{\mathsf{length}(w)}.\,\mathsf{in}\ ext\,]\end{aligned}$$

where w is the word used in the encoding of the machine.

Proof. We distinguish four steps:

(1) When the q_A ambient has been opened, the ambient get_out is liberated and is present within TM:

$$\mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TM}(q_A)\ |\!\} \Longrightarrow \mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TMGetout}\ |\!\}$$

where $\mathtt{TMGetout}$ is the term

$$TM[\,get\_out[\,\mathbf{0}\,]\ \mid\ \mathtt{code}(q_0)\ \mid\dots\mid\ \mathtt{code}(q_n)\ \mid\ \mathtt{tmsoup}\,]\ .$$

(2) This allows the TM ambient to get a get_out 'token', execute the branch containing the out $cell$, and, doing this, liberate a new get_out ambient:

$$\mathtt{WorkRibb}(w_1, w_2)\{\!|\ \mathtt{TMGetout}\ |\!\} \Longrightarrow \mathtt{WorkRibb}(w_1^1\dots w_1^{r-1}, w_1^r.\,w_2)\{\!|\ \mathtt{TMGetout}\ |\!\}$$

Note that the other subterm starting with open get_out could also have been triggered, leading to a blocked state. This is no harm for us, since we want to establish the existence of an execution where the machine exits the ribbon. This way, TM progresses outwards until it is directly inside $ribbon\_left$.

(3) Then TM gets out of $ribbon\_left$, choosing the other branch of $\mathsf{open}\ get\_out$, which leads to the following state:

$$\begin{array}{ll}\mathtt{WorkRibb}(\epsilon, w_1.w_2)\{\!|\ \mathbf{0}\ |\!\} \ \mid\ TM[& cleaner[\,\mathsf{out}\ TM.\,\mathsf{in}\ ribbon\_left\,] \\ & \mid\ coin[\,\mathsf{out}\ TM.\,\mathsf{in}\ ribbon\_left.\,\mathsf{in}\ cell^{\mathsf{length}(w)}.\,\mathsf{in}\ ext\,] \\ & \mid\ \mathsf{code}(q_0)\ \mid \ldots \mid\ \mathsf{code}(q_n)\ \mid\ \mathtt{tmsoup}]\end{array}$$

(4) At this point, the ambient named TM may liberate an ambient $cleaner$ that enters $ribbon\_left$ and starts the cleaning process. TM may also liberate the ambient $coin$ so that we exactly obtain the expected term. □

Remarks 6.12.

- As we already mentioned above, our encoding of the Turing Machine is at this point dependent from the word w that we want it to recognize.
- reason here using $\Longrightarrow$ transitions instead of deterministic reduction $\leadsto$: indeed, we are considering states where the machine has already recognized the word, and we only need to prove that *there exists* some way back to its (*exact*) initial state. This will be enough for the proof of undecidability in Section 6.3.

6.3. **Undecidability of Logical Equivalence.** We can now exploit the encoding we have studied to establish undecidability of $=_L$.

Lemma 6.13 (Loop lemma). *Given a Turing Machine $\mathcal{M}$ and a word w, define the following terms, given from the encoding of $\mathcal{M}$:*

$$\begin{array}{rcl} Q & := & !\mathtt{FrozenRibb}(w)\ \mid\ !\mathtt{OldRibb}\ \mid\ !\mathsf{open}\ msg\ \mid\ !\mathsf{out}\ cell\ \mid\ \mathtt{TMStart}\,, \\ P_0 & := & Q\ \mid\ \mathtt{GrowingRibb}(w) \quad \textit{and} \quad P_1\ :=\ Q\ \mid\ \mathtt{GrowingRibb}(w.\mathit{ff})\,. \end{array}$$

Then $P_0 \Longrightarrow P_1$. Conversely, $P_1 \Longrightarrow P_0$ if and only if the word w may be recognized on a finite (but sufficiently long) ribbon of the shape $w.\mathit{ff}^N$, for some $N \in \mathbb{N}$, by the Turing Machine $\mathcal{M}$.

Proof. The transition $P_0 \Longrightarrow P_1$ follows from Lemma 6.2.

Let us then first assume that w can be recognized on a ribbon of the form $w.\mathit{ff}^N$, that is, w followed by an arbitrary number of ff digits. Then from Lemma 6.2, we can obtain the corresponding extension of the ribbon from state P_1, i.e. exhibit a transition $P_1 \Longrightarrow Q\ \mid\ \mathtt{WorkRibb}(w.\mathit{ff}^N, \epsilon)\{\!|\ \mathbf{0}\ |\!\}\ \mid\ start[\,\mathsf{in}\ TM\,]$. At this point, the ambient $start$ can enter TM and allow it to get into the work ribbon. Then, using the simulation result (Proposition 6.10), we know that the Turing Machine reaches the acceptation state (this result is obtained by induction over the length of w). At this point, according to Lemma 6.11, the work ribbon is transformed into an old ribbon (collected by the corresponding replicated term in Q), the Turing Machine comes out of the ribbon, and waits for a start signal. The liberated $coin$ ambient may progress inside a frozen ribbon (containing word w by definition of Q above) until it reaches the frozen extensor and wakes it up. We then exactly obtain P_0.

Now let us assume that w cannot be recognized on any ribbon. As Q is blocked (in particular, $\mathtt{TMStart}$ is waiting for an ambient $start$ to enter TM), the first reducts of P_1 are of the form $Q\ \mid\ ribbon\_left[\,R\,]$, where $\mathtt{GrowingRibb}(w.\mathit{ff}) \Longrightarrow ribbon\_left[\,R\,]$. If a reduction chain from P_1 to P_0 can be found, then by Lemma 6.2 there exists an integer n such that

$$P_1 \Longrightarrow \underbrace{Q \mid \texttt{WorkRibb}(w.\mathit{ff}^n)\{\!|\, \mathbf{0} \,|\!\} \mid \mathit{start}[\, \mathsf{in}\, TM \,]}_{T} \Longrightarrow P_0 \,.$$

In term T the WorkRibb is blocked, so the only evolution can come from the machine entering a ribbon. We distinguish three cases according to the kind of ribbon which is entered by the machine:

(1) If it gets into an old ribbon, there can be no more reduction, as the TM is stuck on an in *cell* action.
(2) If it gets into the work ribbon, according to Proposition 6.10, there is a unique way to evolve, through simulation of the machine. At this point, the machine may have an infinite computation on the finite ribbon, never reaching accepting state: this means that it will not get out of the ribbon, which prevents the system to evolve into P_0. Alternatively, the machine may try to use more ribbon than what has been created before evolution from GrowingRibb into WorkRibb, and the machine is stuck. So in any case, state P_0 cannot be reached.
(3) We reason similarly in the case where the machine enters a frozen ribbon.

Finally, we have that state P_0 is unreachable if word w cannot be recognised by the machine on a ribbon of the form $w.\mathit{ff}^N$ for some N, which concludes the proof. □

Theorem 6.14 (Undecidability of $=_L$). *$=_L$ is an undecidable relation on MA.*

Proof. Let us first note that the decidability of $=_L$ over $\mathrm{MA_{IF}}$ is a consequence of its inductive characterisation $\sim_{\mathrm{ind}}$ (Definition 4.9) together with the image finitess hypothesis of $\mathrm{MA_{IF}}$.

Consider processes P_0 and P_1 from Lemma 6.13. We show that the problem of deciding whether one can prove $\mathsf{open}\ n.\,P_0 =_L \mathsf{open}\ n.\,P_1$ is equivalent to deciding whether $P_0 \Longrightarrow P_1 \Longrightarrow P_0$. This will be enough, by Lemma 6.13, to obtain the undecidability of $=_L$.

Let us prove now the undecidability of $=_L$ on MA. Consider processes P_0 and P_1 of Lemma 6.13. These processes are in $\mathrm{MA^s_{IF}}$. Using Corollary 4.20, the definition of $\simeq_{\mathrm{int}}$, and Theorem 5.9, we have:

$$\begin{array}{rl} \mathsf{open}\ n.\,P_0 =_L \mathsf{open}\ n.\,P_1 & \text{iff} \quad \mathsf{open}\ n.\,P_0 \ \simeq_{\mathrm{int}}\ \mathsf{open}\ n.\,P_1 \\ & \text{iff} \quad P_0 \Longrightarrow \simeq_{\mathrm{int}}\ P_1 \Longrightarrow \simeq_{\mathrm{int}}\ P_0 \end{array}$$

(from Theorem 5.9, $\Longrightarrow \simeq_{\mathrm{int}}$ is $\Longrightarrow$ on $\mathrm{MA^s_{IF}}$).

The first equivalence follows from soundness and completeness (Theorems 3.29 and 4.19). The second is the definition of $\simeq_{\mathrm{int}}$. Since on $\mathrm{MA^s_{IF}}$ $\simeq_{\mathrm{int}}=\equiv$, the last condition is simply the loop condition, and undecidability follows from Lemma 6.13. □

7. Conclusions and future work

In this paper we have presented a number of characterisations of logical equivalence, including a coinductive characterisation by means of intensional bisimilarity, $\simeq_{\mathrm{int}}$, and an inductive characterisation based on inversion results for $\simeq_{\mathrm{int}}$. These characterisation results are established on the MA calculus in which terms need not be image-finite, and with respect to a finitary logic. We are not aware of other results of this kind. (Characterisation results for a bisimilarity with respect to a modal logic in the literature rely either on an image-finiteness hypothesis for the terms of the language, or on the presence of some infinitary constructs in the syntax of the logic.)

We have compared logical equivalence with barbed congruence, showing that the latter is strictly coarser, and with structural congruence, showing that the two relations are "almost the same" in the (Turing-complete) calculus $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ (the two relations coincide on the synchronous version of $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, whereas an additional eta-law has to be added to structural congruence in the asynchronous calculus). A spin-off of this study is a general better understanding of behavioural equivalences in Ambient-like calculi. For instance, we have shown that behavioural equivalences can be insensitive to stuttering phenomena originated by processes that may repeatedly enter and exit an ambient. Finally, we have proved that logical equivalence, although decidable on $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$, it is not decidable on the whole MA calculus.

We discuss below a few possible extensions of our work. On the logic side, other logical connectives could be added without changing our results, as long as formulas expressing capabilities and replication can still be derived. We believe this holds in particular for the 'somewhere' modality [11], and for fresh quantification [17].

In our work, we have interpreted the 'sometimes' modality ($\Diamond\mathcal{A}$) in a weak sense, which makes intensional bisimilarity a weak form of bisimilarity. We believe that under a strong interpretation of the modality the result corresponding to Theorem 5.9 can be derived in a much simpler way, especially because stuttering does not show up.

On the calculus side, a first variation could be the introduction of a general recursion scheme instead of replication. This would make it possible to express recursion 'in depth', and not only 'in width', as with replication. Our proofs do not obviously carry over to this setting, mainly due to the fact that the sequential degree of a process may then be infinite, and we would lack a measure to reason by induction.

Another interesting extension is the addition of name restriction $(\nu n)P$ to the calculus. Including restriction naturally implies to add its logical counterpart, name revelation ($n®\mathcal{A}$, see [12]) to the logic. Our results can be extended to this setting on the finite calculus, and on infinite processes with only finitely many restricted names, but we do not know how to extend them to richer calculi. For instance, the proof of completeness cannot be directly adapted to the extension with name restriction in the general case. The possibility of generating infinitely many fresh names breaks Lemma 4.12, intuitively because infinitely many frozen subterms can appear as outcomes of a given term. For the same reason, we think that our approach to obtain completeness in absence of an image-finiteness hypothesis cannot be adapted to the π-calculus, where infinitely many names can be generated. However, our results for the $\mathrm{MA}^{\mathrm{s}}_{\mathrm{IF}}$ fragment, in particular Theorem 5.9 ($=_L = \equiv$), still hold in presence of name restriction.

In the paper we have considered only communications of basic names. Certain presentation of the MA calculus also include operators for communication of capabilities. We believe that such communications could be added with mild modifications to the proofs.

References

[1] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998.

[2] I. Boneva and J.-M. Talbot. When ambients cannot be opened. *Theoretical Computer Science*, 333(1-2):127–169, 2005.

[3] N. Busi and G. Zavattaro. On the expressive power of movement and restriction in pure mobile ambients. *Theoretical Computer Science*, 322(3):477–515, 2004.
[4] N. Busi and G. Zavattaro. Deciding Reachability in Mobile Ambients. In *In Proc. of European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 248–262. Springer Verlag, 2005.
[5] L. Caires. Behavioral and Spatial Observations in a Logic for the pi-Calculus. In *Proc. of FOSSACS'04*, volume 2987, pages 72–89. Springer Verlag, 2004.
[6] L. Caires and E. Lozes. Elimination of Quantifiers and Undecidability in Spatial Logics for Concurrency. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*, pages 240–257. Springer Verlag, 2004.
[7] L. Caires and H. Torres Vieira. Extensionality of Spatial Observations in Distributed Systems. *Electr. Notes Theor. Comput. Sci.*, 175(3):131–149, 2007.
[8] C. Calcagno, L. Cardelli, and A. Gordon. Deciding Validity in a Spatial Logic for Trees. In *Proc. of TLDI'03*, pages 62–73. ACM Press, 2003.
[9] L. Cardelli. Describing Semistructured Data. *SIGMOD Record, Database Principles Column*, 30(4), 2001.
[10] L. Cardelli and G. Ghelli. A Query Language Based on the Ambient Logic. In *Proc. of ESOP'01*, volume 2028 of *LNCS*, pages 1–22. Springer Verlag, 2001. invited paper.
[11] L. Cardelli and A. Gordon. Anytime, Anywhere, Modal Logics for Mobile Ambients. In *Proc. of POPL'00*, pages 365–377. ACM Press, 2000.
[12] L. Cardelli and A. Gordon. Logical Properties of Name Restriction. In *Proc. of TLCA'01*, volume 2044 of *LNCS*. Springer Verlag, 2001.
[13] L. Cardelli and A.D. Gordon. Mobile ambients. In *Proc. FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998.
[14] L. Cardelli and A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. 27th POPL*. ACM Press, 2000.
[15] M. Dam. Relevance Logic and Concurrent Composition. In *Proc. of LICS'88*, pages 178–185. IEEE, 1988.
[16] A. Dawar, P. Gardner, and G. Ghelli. Adjunct elimination through games in Static Ambient Logic. In *Proc. of the 24th Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 3328, pages 211–223. Springer Verlag, 2004.
[17] M. Gabbay and A.M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.
[18] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.
[19] D. Hirschkoff. An Extensional Spatial Logic for Mobile Processes. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*, pages 325–339. Springer Verlag, 2004.
[20] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Separability, Expressiveness and Decidability in the Ambient Logic. In *17th IEEE Symposium on Logic in Computer Science*, pages 423–432. IEEE Computer Society, 2002.
[21] D. Hirschkoff, E. Lozes, and D. Sangiorgi. On the expressiveness of the ambient logic. *Logical Methods in Computer Science*, 2(2), 2006.
[22] D. Hirschkoff and D. Pous. A distribution law for ccs and a new congruence result for the *pi*-calculus. In *Proc. of FoSSaCS'07*, volume 4423 of *Lecture Notes in Computer Science*, pages 228–242, 2007.
[23] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.
[24] F. Levi and D. Sangiorgi. Controlling interference in ambients. Short version appeared in *Proc. 27th POPL*, ACM Press, 2000.
[25] F. Levi and D. Sangiorgi. Mobile Safe Ambients. *ACM Trans. Program. Lang. Syst.*, 25(1):1–69, 2003. Short version appeared in *Proc. 27th POPL*, ACM Press.
[26] E. Lozes. Elimination of spatial connectives in static spatial logics. *Theoretical Computer Science*, 330(3):475–499, 2005.
[27] S. Maffeis and I. Phillips. On the Computational Strength of Pure Ambient Calculi. *Theoretical Computer Science*, 330(3):501–551, 2005.
[28] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
[29] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proc. 19th ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer Verlag, 1992.

[30] D. Sangiorgi. Extensionality and Intensionality of the Ambient Logic. In *Proc. of 28th POPL*, pages 4–17. ACM Press, 2001.
[31] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
[32] S. Dal Zilio. Structural Congruence for Ambients is Decidable. In *Proc. of ASIAN'00*, volume 1961 of *LNCS*. Springer Verlag, 2000.