Separability, expressiveness, and decidability in the Ambient Logic

Daniel Hirschkoff — ENS Lyon
Etienne Lozes — ENS Lyon
Davide Sangiorgi — INRIA

January 15, 2002

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.

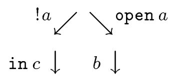
We study some basic questions concerning the descriptive and discriminating power of AL, focusing on the equivalence on processes induced by the logic ($=_L$). We consider MA, and two Turing complete subsets of it, MA$_{\text{IF}}$ and MA$^{\text{syn}}_{\text{IF}}$, respectively defined by imposing a semantic and a syntactic constraint on process prefixes.

The main contributions include: coinductive and inductive operational characterisations of $=_L$; an axiomatisation of $=_L$ on MA$^{\text{syn}}_{\text{IF}}$; the construction of characteristic formulas for the processes in MA$_{\text{IF}}$ with respect to $=_L$; the decidability of $=_L$ on MA$_{\text{IF}}$ and on MA$^{\text{syn}}_{\text{IF}}$, and its undecidability on MA.

1 Introduction

The *Ambient Logic*, AL, [CG00] is a modal logic for expressing properties of processes in the calculus of Mobile Ambients, MA [CG98, CG99]. In MA the unit of movement is an ambient, which, intuitively, is a named location. An ambient may contain other ambients, and *capabilities*, which determine the ambient movements. The primitives for movement allow: an ambient to enter a sibling ambient; an ambient to exit the parent ambient; a process to dissolve an ambient boundary. MA has a replication operator to make a process persistent, that is, to make infinite copies of the process available.

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.[1] The trees are unordered: the order of the children of a node is not important. As an example, the process $P \overset{\text{def}}{=} !a[\mathtt{in}\,c] \mid \mathtt{open}\,a.\,b[\mathbf{0}]$ is represented by the tree:



The replication $!a$ indicates that the resource $a[\mathtt{in}\,c]$ is persistent: unboundedly many such ambients can be spawned. By contrast, $\mathtt{open}\,a$ 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 $\mathtt{in}\,c \mid !a[\mathtt{in}\,c] \mid b[\mathbf{0}]$. However, the process $!a[\mathtt{in}\,c] \mid !\mathtt{open}\,a.\,b[\mathbf{0}]$ can evolve into any process of the form

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

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*.

---

[1] We are using a tree representation different from that of Cardelli and Gordon, but more convenient to our purposes.

In summary, MA is a calculus of dynamically-evolving unordered edge-labelled trees. AL is a logic for reasoning on such trees. Indeed, the actual definition of satisfaction of the formulas is given on MA processes quotiented by a relation of *structural congruence*, which equates processes with the same tree representation. (This relation is similar to Milner's structural congruence for the π-calculus [Mil91].)

AL has also been advocated as a foundation of query languages for semistructured data [CG01a]. 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 and standard models of semistructured data.

AL has a connective that talks about *time*, that is, how processes can evolve. The logic has also connectives that talk about *space*, that is, the shape of the edge-labelled trees that describe process distributions. AL is quite different from standard modal logics. First, such 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 [HM85], 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 some basic questions concerning the descriptive and discriminating power of AL. We consider, besides the calculus MA, two subsets of it, obtained by imposing constraints on the processes underneath capabilities. In $\mathrm{MA_{IF}}$, these processes must be image-finite; in $\mathrm{MA_{IF}^{syn}}$, they must be finite. These definitions might appear ad hoc, but they express precisely the constraints needed in some of our results. A further interest of $\mathrm{MA_{IF}^{syn}}$ is that its definition is purely syntactic. Both $\mathrm{MA_{IF}}$ and $\mathrm{MA_{IF}^{syn}}$ are Turing complete, and contain processes that are not image-finite.

We describe the main contributions of the paper. We write $=_L$ to indicate the process equivalence induced by the logic, whereby two terms are equated if they satisfy the same sets of formulas. First, we exhibit two operational characterisations of $=_L$ on MA, which do not mention the logic. Characterisations of the equivalence of a logic allow us to understand the notion of equality on processes – a fundamental notion in process calculi – induced by the logic. One characterisation is coinductive, as a form of labelled bisimilarity. The other is inductive, and uses a well-founded measure on the structure of processes.

Second, we prove that $=_L$ coincides with structural congruence on $\mathrm{MA_{IF}^{syn}}$. This gives us an axiomatisation of $=_L$ on $\mathrm{MA_{IF}^{syn}}$. This axiomatic characterisation is false on the larger class $\mathrm{MA_{IF}}$.

Our third contribution is the construction of characteristic formulas for equivalence classes for $=_L$ in $\mathrm{MA_{IF}}$. We define, for any process $P \in \mathrm{MA_{IF}}$, a formula $\mathcal{F}_P$ such that $Q \models \mathcal{F}_P$ holds iff $Q =_L P$, for all $Q \in \mathrm{MA}$. The result shows that we can talk about the discriminating power of the logic from within the logic itself, at least under some image-finiteness conditions. A corollary is the undecidability of the model-checking problem on $\mathrm{MA_{IF}^{syn}}$ and richer calculi.

Our fourth contribution is on (un)decidability. As a consequence of the inductive characterisation of $=_L$, we can prove that $=_L$ is decidable on $\mathrm{MA_{IF}}$ and $\mathrm{MA_{IF}^{syn}}$. However, if we drop the image-finiteness conditions of $\mathrm{MA_{IF}}$, then $=_L$ becomes undecidable. We show this via an encoding of the halting problem of Turing Machines. The encoding of Turing Machines is actually in $\mathrm{MA_{IF}^{syn}}$, which is thus proved to be Turing Complete. This result is not in contradiction with the decidability of $=_L$ in $\mathrm{MA_{IF}}$ and $\mathrm{MA_{IF}^{syn}}$, because the encoding is correct for reductions but not behaviourally (the process encoding a machine and its derivatives do not need to be in the relation $=_L$).

Most of the results mentioned above are rather different from the usual results of modal logics. Typically, the definition of characteristic formulas exploits fixed-point operators, and the characterised processes are finite-state [GS86, SI94]. AL, by contrast, has no fixed point operator; moreover the image-finiteness condition on processes is weaker than finite-state. ('Image-finite' expresses finiteness on internal reductions, whereas 'finite-state' also takes into account computations containing visible actions such as input and output actions.)

Also, coinductive characterisations of an inductive relation or of the equivalence of a logic usually rely on either image-finiteness of the processes, or on some infinitary operator of the logics, such as infinite conjunctions. In our case we need none of these hypotheses. Further, the inductive relation is not the stratification of the coinductive relation [Mil89], but uses a structural measure on pro-

							Processes
$a, b, \ldots, n, m,$			*Names*	P, Q, R	$::=$	$\mathbf{0}$	*(nil)*
			Capabilities		$\mid$	$P \mid Q$	*(parallel)*
cap	$::=$	$\mathsf{in}\, n$	*(enter)*		$\mid$	$\mathsf{cap}.P$	*(prefixing)*
	$\mid$	$\mathsf{out}\, n$	*(exit)*		$\mid$	$n[P]$	*(ambient)*
	$\mid$	$\mathsf{open}\, n$	*(open)*		$\mid$	$!P$	*(replication)*

Table 1: The syntax of finite MA

---

cesses. Finally, in process calculi decidability is usually unrelated to image-finiteness: for instance, the transition relation of the π-calculus is image-finite, yet strong bisimilarity is undecidable [SW01].

Also the actual form of image-finiteness that we use is non-standard. Behavioural equivalence in MA is insensitive to *stuttering* phenomena, originated by processes that may repeatedly enter and exit an ambient. As a consequence, a computation in which all visible actions are stuttering is semantically equivalent to an internal reduction.

In the proofs of the results two groups of technical lemmas are important. The first group is about the construction of formulas for describing all forms of labels of the trees of MA. The formulas for the replicated labels give us (some of) the power of the ! operator ('of course') of linear logic; this was somehow unexpected, because AL has no infinitary operators, or operators that talk about resources with infinite multiplicity. (We obtain only *some* of this power, because we have to impose constraints on the replicated formulas.)

Other useful formulas that we have derived are the following: a formula ϕ_{fin} that characterises the ephemeral processes (that is, $P \models \phi_{\mathsf{fin}}$ iff the tree of P has no replicated labels); for any set $\mathcal{S}$ of names, a formula $\mathsf{refers}\ \mathcal{S}$ that characterises the processes whose set of free names is precisely $\mathcal{S}$.

The second group of technical lemmas captures decomposition properties of processes. For instance, Lemma 3.6 shows that any two processes P and Q in the relation $=_L$ admit decompositions $P = \mathcal{C}[\widetilde{P}]$ and $Q = \mathcal{D}[\widetilde{Q}]$ where the contexts $\mathcal{C}$ and $\mathcal{D}$ are of a certain syntactic form, and, moreover, both the contexts $\mathcal{C}$ and $\mathcal{D}$, and the continuations $\widetilde{P}$ and $\widetilde{Q}$ (more precisely, processes obtained from appropriate transformation of these) are equivalent.

There are strong connections among all the results discussed above. For instance, both the characterisations of $=_L$ and the characteristic formulas talk about the separability power of AL. The connections are explicit in the proofs: for instance, the proof of undecidability relies on most of the other results.

Related work. Characterisations and axiomatisations of $=_L$ have already been presented in [San01], on *finite* MA (without replications). The proofs rely on the ephemeral nature of the processes, precisely on the property that all (complete) computations of a process, comprising its interactions with the environment, are finite and terminate with the $\mathbf{0}$ process. Therefore we could not adapt these proofs to processes with replications. The need for stuttering in MA is already pointed out in [San01], but all examples use trees with unbounded depth.

We are not aware of other axiomatisations of semantic equivalences in non-finite higher-order process calculi, and of characteristic formulas for logics for mobile processes. Formulas in AL, or similar logics, that characterise the free names of processes were known [CG01b, CC01], but use additional operators (notably the *revelation* operator). The undecidability of the model-checking problem of AL – in fact of an even smaller logic – had already been established, using different techniques [CT01].

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

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

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}\ \texttt{Red-Par} \qquad \frac{P \longrightarrow P'}{n[P] \longrightarrow n[P']}\ \texttt{Red-Amb}$$

$$\frac{P \equiv P' \qquad P' \longrightarrow P'' \qquad P'' \equiv P'''}{P \longrightarrow P'''}\ \texttt{Red-Str}$$

Table 2: The rules for reduction

2 Background

We recall here the syntax of 'pure' MA, from [CG98]. ('Pure' means that computation is only movement; there are no communications.) Also, as in [CG00, Car99, CG01a], the calculus has no restriction operator for creating new names. The restriction-free calculus is simpler, and has a more direct correspondence with edge-labelled trees and semistructured data. Table 1 presents the syntax. The set of names is infinite. *Capabilities* are ranged over by cap, processes by P, Q, R, S. Processes with the same internal structure are identified. This is expressed by means of the *structural congruence relation*, $\equiv$, the smallest congruence such that $(\mathbf{0}, \mid, !)$ is a multiset algebra, that is, satisfies the following rules:

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

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

Relation $\equiv$ is decidable on MA, as well as in other calculi such as the π-calculus [DZ00, EG99]. The rules for the reduction relation, $\longrightarrow$, are given in Table 2. The reflexive and transitive closure of $\longrightarrow$ is written $\Longrightarrow$.

Definition 2.1 (Labelled transitions and stuttering) *We write:*

- $P \overset{\mathsf{cap}}{\longrightarrow} P'$, *if* $P \equiv \mathsf{cap}.\, P_1 \mid P_2$ *and* $P' = P_1 \mid P_2$.
- **(stuttering)** $P \xRightarrow{(M_1, M_2)^\star} P'$ *if there are* $r \geq 1$ *and processes* $P_1, \ldots, P_r$ *with* $P = P_1$ *such that* $P_r \Longrightarrow \overset{M_1}{\longrightarrow} \Longrightarrow \overset{M_2}{\longrightarrow} \Longrightarrow P_{r+1}$ *for all* $1 \leq i < r$, *and* $P_r \Longrightarrow P'$.
- *Finally,* $\overset{\langle \mathsf{cap} \rangle}{\Longrightarrow}$ *is a convenient notation for compacting statements involving capability transitions.* $\overset{\langle \mathsf{in}\ n \rangle}{\Longrightarrow}$ *is* $\xRightarrow{(\mathsf{out}\ n, \mathsf{in}\ n)^\star}$; *similarly* $\overset{\langle \mathsf{out}\ n \rangle}{\Longrightarrow}$ *is* $\xRightarrow{(\mathsf{in}\ n, \mathsf{out}\ n)^\star}$; *and* $\overset{\langle \mathsf{open}\ n \rangle}{\Longrightarrow}$ *is* $\Longrightarrow$.

Some of our results are proved by induction on the *sequentiality degree* of a process, which is the maximal depth of nesting of capabilities in the process, and exploit the following lemma:

Lemma 2.2 *If* $P \longrightarrow Q$ *or* $P \overset{\mathsf{cap}}{\longrightarrow} Q$ *then the sequentiality degree of* P *is greater than, or equal to, that of* Q.

The logic. To define the set of formulas of the Ambient Logic (AL) we introduce an infinite set of *variables*, ranged over with x, y, z; η ranges over names and variables. 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, $\Diamond\mathcal{A}$ has been briefly discussed in the Introduction. 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 $\mathcal{A}@\eta$ are the logical adjuncts of $\mathcal{A} \mid \mathcal{B}$ and $\eta[\mathcal{A}]$, in the sense of being, roughly, their inverse (see below). A formula without free variables is *closed.*

Definition 2.3 *The* satisfaction *relation is defined on closed formulas as follows:*

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

The logic in [CG00] has also a *somewhere* connective that holds of a process containing, at some arbitrary level of nesting of ambients, an ambient whose content satisfies $\mathcal{A}$. The addition of this connective would not change the results in the paper.

We give $\vee$ and $\wedge$ the least syntactic precedence, thus $\mathcal{A}_1 \triangleright \mathcal{A}_2 \wedge \mathcal{A}_3$ reads $(\mathcal{A}_1 \triangleright \mathcal{A}_2) \wedge \mathcal{A}_3$, and $\mathcal{A}_1 \triangleright (\Diamond \mathcal{A}_2 \wedge \Diamond \mathcal{A}_3)$ reads $\mathcal{A}_1 \triangleright ((\Diamond \mathcal{A}_2) \wedge (\Diamond \mathcal{A}_3))$. We shall use the dual of some connectives, namely the duals of linear implication ($\mathcal{A} \blacktriangleright \mathcal{B}$), of the sometime modality ($\Box \mathcal{A}$), of the parallel operator ($\|$), and the standard duals of universal quantification ($\exists\, x\,.\, \mathcal{A}$) and disjunction ($\mathcal{A} \wedge \mathcal{B}$). We also define (classical) implication ($\mathcal{A} \rightarrow \mathcal{B}$).

$$\begin{array}{rcl rcl rcl}
\mathcal{A} \blacktriangleright \mathcal{B} & \overset{\text{def}}{=} & \neg(\mathcal{A} \triangleright \neg \mathcal{B}) & \Box \mathcal{A} & \overset{\text{def}}{=} & \neg \Diamond \neg \mathcal{A} & \mathcal{A} \| \mathcal{B} & \overset{\text{def}}{=} & \neg(\neg \mathcal{A} \mid \neg \mathcal{B}) \\
\exists\, x\,.\, \mathcal{A} & \overset{\text{def}}{=} & \neg \forall\, x\,.\, \neg \mathcal{A} & \mathcal{A} \wedge B & \overset{\text{def}}{=} & \neg(\neg \mathcal{A} \vee \neg \mathcal{B}) & \mathcal{A} \rightarrow \mathcal{B} & \overset{\text{def}}{=} & \neg \mathcal{A} \vee \mathcal{B}
\end{array}$$

Thus $P \models \mathcal{A} \blacktriangleright \mathcal{B}$ iff there exists Q with $Q \models \mathcal{A}$ and $P \mid Q \models \mathcal{B}$, and $P \models \Box \mathcal{A}$ iff $P' \models \mathcal{A}$ for all P' such that $P \Longrightarrow P'$. The formula $\mathcal{A}^{\forall} \overset{\text{def}}{=} \mathcal{A} \| \neg \top$, from [CG00], is satisfied by P iff for any Q, R such that $P \equiv Q \mid R$, it holds that $Q \models \mathcal{A}$.

Definition 2.4 (Process logical equivalence) *For processes P and Q, we write $P =_L Q$ if for all closed formulas $\mathcal{A}$ it holds that $P \models \mathcal{A}$ iff $Q \models \mathcal{A}$.*

3 Coinductive and inductive operational relations

The coinductive relation below follows the definition of intensional bisimilarity in finite MA [San01].

Definition 3.1 (Intensional bisimilarity) Intensional bisimilarity *is the largest symmetric relation $\simeq_{\text{bis}}$ on processes such that $P \simeq_{\text{bis}} Q$ 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 $P_i \simeq_{\text{bis}} Q_i$, 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 $P' \simeq_{\text{bis}} Q'$.*
4. *For any cap, if $P \overset{\mathsf{cap}}{\longrightarrow} P'$ then there is Q' such that $Q \overset{\mathsf{cap}}{\longrightarrow} \overset{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'$ and $P' \simeq_{\text{bis}} Q'$.*
5. *If $P \equiv n[P']$ then there is Q' such that $Q \equiv n[Q']$ and $P' \simeq_{\text{bis}} Q'$.*

With respect to standard bisimilarities for process calculi, $\simeq_{\text{bis}}$ has intensional clauses, namely (1), (2) and (5), which allow us to observe parallel compositions, terminated process, and ambients. These clauses correspond to the intensional connectives '|', '0', and $n[\mathcal{A}]$ of the logic. The other main peculiarity of $\simeq_{\text{bis}}$ are the stuttering relations. The need for stuttering on infinite trees (i.e., MA with a recursion operator) has been pointed out in [San01]. We show that stuttering is also needed on finite trees with replication.

Example 1 *Consider the processes* $P_0 \overset{\text{def}}{=} !\mathsf{open}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}] \mid n[\mathbf{0}]$, *and* $P_1 \overset{\text{def}}{=} !\mathsf{open}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}] \mid \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}]$. *It holds that* $P_0 \not\simeq_{\text{bis}} P_1$*; however, since*

$$P_0 \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^\star} P_1 \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^\star} P_0\,,$$

we have $\mathsf{out}\, n.\, P_0 \;\simeq_{\text{bis}}\; \mathsf{out}\, n.\, P_1$. *Without stuttering this equivalence would not hold.*

The proof of congruence of $\simeq_{\text{bis}}$ follows the proof of the analogous result in [San01], using a technique similar to Howe's for proving congruence of bisimilarity in higher-order languages [How96].

Theorem 3.2 (Soundness of $\simeq_{\text{bis}}$) *In MA,* $\simeq_{\text{bis}} \subseteq =_L$.

Completeness – the hard implication – is proved in Section 4. Below we show an inductive characterisation of $\simeq_{\text{bis}}$. The crux of this result is Lemma 3.4, which gives us a decomposition property for bisimilar processes in terms of special forms of contexts.

Definition 3.3 (Contexts, active contexts, and coloured contexts) *A* context *(ranged over with* $\mathcal{C}, \mathcal{D}$*) is a process term with some holes* $[\,]_i$ *in it, each hole occurring once. A context is* active *if each hole appears underneath exactly one capability. A* coloured context $\mathcal{C}(\sigma)$ *is given by an active context* $\mathcal{C}$ *and a* colouring function σ*, assigning a colour to each hole (we assume there are infinitely many colours available).*

Structural congruence is defined on coloured contexts as on processes, but with the additional rule for holes saying that $[\,]_i \equiv [\,]_j$ if the two holes have the same colour.

Lemma 3.4 (Decomposition lemma) *Suppose* $P \simeq_{\text{bis}} Q$*. Then there are two active contexts* $\mathcal{C}$ *and* $\mathcal{D}$*, and two vectors of processes* $\widetilde{P}, \widetilde{Q}$ *such that* $P \equiv \mathcal{C}[\widetilde{P}], Q \equiv \mathcal{D}[\widetilde{Q}]$ *and there is a colouring function* σ *such that*

1. $\mathcal{C}(\sigma) \;\equiv\; \mathcal{D}(\sigma)$,
2. *for any* i, j*, if* $[\,]_i$ *and* $[\,]_j$ *have the same colour, then they are underneath the same capability* cap_i*. Moreover, there are* P'_i, Q'_j *such that* $P_i \overset{\mathsf{cap}_i}{\Longrightarrow} P'_i \simeq_{\text{bis}} Q_j$*, and* $Q_j \overset{\mathsf{cap}_i}{\Longrightarrow} Q'_j \simeq_{\text{bis}} P_i$.

In the lemma, subcomponents $\widetilde{P}$ and $\widetilde{Q}$, and their derivatives P'_i, Q'_j, have a sequentiality degree (that is, the depth of nesting of capabilities) strictly smaller than that of the original processes P and Q (for the derivatives, this is given by Lemma 2.2). We can therefore exploit this result to obtain an inductive characterisation of $\simeq_{\text{bis}}$. Let $\sim_{\text{ind}}$ be this inductive relation, i.e. $\sim_{\text{ind}}$ is the least relation such that whenever two active contexts $\mathcal{C}, \mathcal{D}$ and two vectors of processes $\widetilde{P}$ and $\widetilde{Q}$ satisfy the clause of the lemma, where $\simeq_{\text{bis}}$ is replaced by $\sim_{\text{ind}}$, then also $P \sim_{\text{ind}} Q$, for all processes P, Q with $P \equiv \mathcal{C}[\widetilde{P}]$, $Q \equiv \mathcal{D}[\widetilde{Q}]$.

Theorem 3.5 *Relations* $\simeq_{\text{bis}}$ *and* $\sim_{\text{ind}}$ *coincide on MA.*

Lemma 3.4, and therefore also the definition of $\sim_{\text{ind}}$, contain a heavy hidden universal quantification, due to the decompositions of P and Q up to $\equiv$, for an equivalence class of $\equiv$ contains an infinite number of processes. The following lemma shows however that the active-context decomposition of a process is essentially unique.

Lemma 3.6 (Uniqueness of active context decomposition) *Suppose there are processes* $P, \widetilde{P_i}$*, and active contexts* $\mathcal{C}_i$ $(i = 1, 2)$*, such that* $P \equiv \mathcal{C}_i[\widetilde{P_i}]$*. Then there is a colouring function* σ *such that*

1. $\mathcal{C}_1(\sigma) \;\equiv\; C_2(\sigma)$;
2. *for any* i, j*, if* $[\,]_i$ *and* $[\,]_j$ *have the same colour, then* $P_i \;\equiv\; Q_j$*, and the two holes are underneath the same capability.*

We define some subclass of processes, used in the statements of some of our results.

Definition 3.7 (image-finite processes, finite processes, $\mathbf{MA}_{\mathrm{IF}}$, $\mathbf{MA}_{\mathrm{IF}}^{\mathrm{syn}}$) *A process P is* image finite *if the set $\{P' \mid P \overset{\langle cap \rangle}{\Longrightarrow} P'\}$, quotiented by $\simeq_{\mathrm{bis}}$, is finite. P is* finite *if there is Q such that $P \equiv Q$ and Q has no replication operator. The classes MA_{IF} and $MA_{\mathrm{IF}}^{\mathrm{syn}}$ are obtained by adding to the grammar of Table 1 the following constraints on the production $\mathsf{cap}.P$: in MA_{IF}, the continuation P must be an image-finite process of MA_{IF}; in $MA_{\mathrm{IF}}^{\mathrm{syn}}$, the continuation P must be finite.*

All the above classes of processes are closed under transition. The inclusions $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}} \subseteq \mathrm{MA}_{\mathrm{IF}} \subseteq \mathrm{MA}$ are strict. The processes P_0 and P_1 in Example 1 are in $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$ and image-finite; however $\mathsf{out}\ n.P_0$ and $\mathsf{out}\ n.P_1$ are in $\mathrm{MA}_{\mathrm{IF}}$ but not in $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$. The process $P = \mathsf{open}\ n.(!\mathsf{open}\ a \mid !a[b[\mathbf{0}]])$ is in MA, but not in $\mathrm{MA}_{\mathrm{IF}}$, because $!\mathsf{open}\ a \mid !a[b[\mathbf{0}]]$ is not image finite.

$\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$, and hence also $\mathrm{MA}_{\mathrm{IF}}$, contains processes that are not image-finite: for instance, the processes used to encode Turing Machines in Section 5.

Another consequence of Theorem 3.5 is:

Theorem 3.8 (Characterisation of $\simeq_{\mathrm{bis}}$ on $\mathbf{MA}_{\mathrm{IF}}^{\mathrm{syn}}$) *Let $P \in MA_{\mathrm{IF}}^{\mathrm{syn}}$. Then for all $Q \in MA$, $P \simeq_{\mathrm{bis}} Q$ iff $P \equiv Q$.*

The equality $\mathsf{out}\ n.P_0 \simeq_{\mathrm{bis}} \mathsf{out}\ n.P_1$ of Example 1 shows that Theorem 3.8 cannot be extended to $\mathrm{MA}_{\mathrm{IF}}$: the $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$ requirement of finiteness for the processes underneath capabilities seems essential for the theorem.

4 Characteristic formulas and completeness

A characteristic formula of a process P is a formula that is satisfied by all and only the processes Q in the relation $\simeq_{\mathrm{bis}}$ with P. In this section we derive characteristic formulas for the processes in $\mathrm{MA}_{\mathrm{IF}}$.

An MA process can be viewed as a finite labelled tree in which labels can be ambient names, capabilities, replicated ambients, and replicated capabilities. If we can define formulas that describe all these labels, then we will be able to derive the characteristic formulas using standard techniques for image-finite processes with finite tree representation [GS86, SI94]. AL has formulas $n[\mathcal{A}]$ that talk about ambient labels. We have to construct the formulas for the other labels. Formulas for capabilities are presented in [San01], for finite MA; they are also correct on MA:

Lemma 4.1 (Formulas for capabilities) *There is a computable function that associates to each capability cap and formula $\mathcal{A}$ a formula $\langle \mathsf{cap} \rangle.\mathcal{A}$ such that $P \models \langle \mathsf{cap} \rangle.\mathcal{A}$ iff there are P' and P'' such that $P \equiv \mathsf{cap}.P'$ and $P' \overset{\langle cap \rangle}{\Longrightarrow} P''$ with $P'' \models \mathcal{A}$.*

A formula $\langle \mathsf{cap} \rangle.\mathcal{A}$ expresses possibility (in Lemma 4.1, *at least one* derivative of P' satisfies $\mathcal{A}$). We also need formulas $[\![\mathsf{cap}]\!].\mathcal{A}$ for *necessity* (*all* derivatives of P' satisfy $\mathcal{A}$). Such formulas are not the dual of the possibility formulas, as in standard modal logics, because of the spatial aspects of AL. For instance, $[\![\mathsf{in}\ n]\!].\top$ is different from $\neg\langle \mathsf{in}\ n \rangle.\neg\top$: the latter is also satisfied by $n[P] \mid m[Q]$, for any n, m, P, Q. We set:

$$[\![\mathsf{cap}]\!].\mathcal{A} \overset{\mathrm{def}}{=} \langle \mathsf{cap} \rangle.\mathcal{A} \wedge \neg\langle \mathsf{cap} \rangle.\neg\mathcal{A}\,.$$

The challenging part, however, is the definition of replicated formulas $!\mathcal{A}$ with the property

$$\begin{array}{l} P \models !\mathcal{A} \text{ iff there are } r > 1, s \geq r, P_i\ (1 \leq i \leq s) \text{ such that} \\ P \equiv \Pi_{1 \leq i \leq r} !P_i \mid \Pi_{r+1 \leq i \leq s} P_i\,, \text{ and } P_i \models \mathcal{A} \text{ for all } 1 \leq i \leq s\ , \end{array} \qquad (1)$$

where $\Pi_{1 \leq i \leq t} Q_i$ abbreviates $Q_1 \mid \ldots \mid Q_t$.

We say that a component of a process is *at top level* if the component is not underneath a capability or inside an ambient. A process is *single* if it is structurally congruent to a process of the form $n[P]$ or $\mathsf{cap}.P$. The formula $\mathsf{1Comp} \overset{\mathrm{def}}{=} 0 \| 0\ \wedge\ \neg 0$ (from [San01]) characterises the single processes.

The definition of $!\mathcal{A}$ has two parts. The first part says that if $P \models \mathcal{A}$ then all parallel components in P that are single and at top level satisfy $\mathcal{A}$. This is expressed by the formula

$$\mathcal{A}^{\omega} \overset{\mathrm{def}}{=} (\mathsf{1Comp} \rightarrow \mathcal{A})^{\forall}\,.$$

The second part of the definition of $!\mathcal{A}$ addresses persistence, by saying that there are infinitely many processes at top level that satisfy $\mathcal{A}$. We have to say that these infinite copies are at top level: for instance, $!\langle \mathsf{cap} \rangle .\, \mathbf{0}$ (or, in fact, any other replicated formula) should not be satisfied by $\mathsf{cap}.\, !\mathsf{cap}.\, \mathbf{0}$. We can talk about the top level because we can express in the logic the maximal depth of nesting of capabilities (the sequentiality depth, Section 2) and the maximal depth of nesting of ambients in a process. Indeed, a component of P is at the top level iff it has the same depths of nesting as P. As a consequence, however, we have to impose a constraint on the definition of $!\mathcal{A}$: all processes that satisfy $\mathcal{A}$ should have the same depths of nesting. We say that these formulas have *fixed model depth.*

The other constraint we need on $\mathcal{A}$ roughly requires that all the processes that satisfy $\mathcal{A}$ should be single and have the same outermost operator. Precisely, either $P \models \mathcal{A}$ should imply $P \equiv \mathsf{cap}.\, P'$, for some cap, P' (in this case, $\mathcal{A}$ is *single for* cap); or $P \models \mathcal{A}$ should imply $P \equiv n[P']$, for some n, P' (then $\mathcal{A}$ is *single for* n). Moreover, the construction of $!\mathcal{A}$ depends on such outermost operator.

The definition of $!\mathcal{A}$ is in Appendix A. Here we only show an example:

$$!\llbracket \mathsf{open}\ n \rrbracket .\, \mathcal{A} \stackrel{\text{def}}{=} (\llbracket \mathsf{open}\ n \rrbracket .\, \mathcal{A})^{\omega} \ \wedge \ (n[0])^{\omega} \rhd \Box\, (\llbracket \mathsf{open}\ n \rrbracket .\, \mathcal{A} \ \mid \ \top\ ) .$$

Lemma 4.2 *For each capability* cap *there is a computable function that associates to each formula* $\mathcal{A}$ *that is single for* cap *and has fixed model depth, a formula* $!\mathcal{A}$ *with the property (1).*

The assertion for the formulas single for names is similar.

We show some concrete examples of characteristic formulas. The general definitions are in Appendix A. A characteristic formula for $!\mathsf{open}\ n.\, n[\mathbf{0}]$ is $!\llbracket \mathsf{open}\ n \rrbracket .\, n[\mathbf{0}]$. A characteristic formula for $!\mathsf{open}\ n.\, (\mathsf{open}\ n \mid n[\mathbf{0}])$ is

$$\mathcal{F}_1^{\omega} \ \wedge \ (n[0])^{\omega} \rhd \Box\, (\mathcal{F}_1 \ \mid \ \top) \quad \text{where} \quad \begin{cases} \mathcal{F}_1 \stackrel{\text{def}}{=} \langle \mathsf{open}\ n \rangle .\, \mathcal{F}_2 \wedge \llbracket \mathsf{open}\ n \rrbracket .\, (\mathcal{F}_2 \vee 0) \\ \mathcal{F}_2 \stackrel{\text{def}}{=} \llbracket \mathsf{open}\ n \rrbracket .\, 0 \mid n[0] \end{cases}$$

and $\mathcal{F}_2$ is a characteristic formula for $\mathsf{open}\ n \mid n[\mathbf{0}]$.

Theorem 4.3 (Characteristic formulas for $\mathbf{MA}_{\mathrm{IF}}$) *There is a computable function that associates to each* $P \in MA_{\mathrm{IF}}$ *a formula* $\mathcal{F}_P$ *such that for any* $Q \in MA$,

$$Q \models \mathcal{F}_P \qquad \textit{iff} \qquad P \simeq_{\mathrm{bis}} Q.$$

We shall see that $\simeq_{\mathrm{bis}}$ coincides with $=_L$, thus the result can also be formulated in terms of characteristic formulas for $=_L$.

For some of the constructions above we use some special formulas that are of independent interest. One such formula is satisfied by precisely the finite processes. It is derived by exploiting the lemma below, which gives us an operational characterisation of 'finiteness'.

Lemma 4.4 $P \in MA$ *is finite iff there are* Q, R, n *such that* $n[P \mid Q] \mid R \Longrightarrow \mathbf{0}$.

Proposition 4.5 *Let* $\phi_{\mathsf{fin}} \stackrel{\text{def}}{=} \exists x.\, \big(\top \blacktriangleright (\top \blacktriangleright \Diamond 0)@x\big)$. *For any* $P \in MA$, $P \models \phi_{\mathsf{fin}}$ *iff* P *is finite.*

We can also define, for any finite set $\mathcal{S}$ of names, a formula satisfied by those processes whose set of free names is precisely $\mathcal{S}$. For this construction we exploit the ability, using the modal formulas for capabilities, to detect unguarded occurrences of names, together with Lemma 4.6. A process P is *flat* if the only process underneath all capabilities and inside all ambients of P is $\mathbf{0}$.

Lemma 4.6 *For all* P, n, $n \in \mathrm{fn}(P)$ *iff for any name* m, *there are some flat processes* Q, R, *and a process* S *with an occurrence of* n *at top level such that* $m[P \mid Q] \mid R \Longrightarrow m[S]$.

Proposition 4.7 *There is a computable function that associates to each finite set* $\mathcal{S}$ *of names a formula* $\mathsf{refers}\ S$ *such that for any* $P \in MA$

$$P \models \mathsf{refers}\, \mathcal{S} \qquad \textit{iff} \qquad \mathcal{S} = \mathrm{fn}(P)\,.$$

The definition of $\mathsf{refers}\ \mathcal{S}$ is in Appendix A.

We derive the completeness of $\simeq_{\mathrm{bis}}$ by exploiting the formulas above, and two further lemmas. The first lemma essentially says that we can construct characteristic formulas for any active context

(Definition 3.3). Its actual statement is rather technical, and we omit it. We explain the other key result, Lemma 4.8. A process in MA may be non-image-finite; indeed, the set $\mathcal{K}$ of derivatives of a process may be infinite even after quotienting $\mathcal{K}$ with $\simeq_{\mathrm{bis}}$. Such forms of infinity are a major technical problem for non-coinductive characterisations of a bisimilarity. In the case of $\simeq_{\mathrm{bis}}$ we have been able to overcome it by using a weaker form of finiteness: the set of processes in $\mathcal{K}$ with an active context that is 'equivalent' to a given context $\mathcal{C}$, quotiented by $\simeq_{\mathrm{bis}}$, is finite.

Lemma 4.8 *Let $P \in MA$ and $\mathcal{C}$ an active context. Call $\mathcal{E}$ the set of processes P' such that $P \Longrightarrow P'$ and there exists a decomposition $P' \equiv \mathcal{C}'[\widetilde{P'}]$ of P' and a colouring function σ such that $\mathcal{C}(\sigma) \equiv \mathcal{C}'(\sigma)$. Then $\mathcal{E}$, quotiented by $\simeq_{\mathrm{bis}}$, is finite.*

Theorem 4.9 (Completeness of $\simeq_{\mathrm{bis}}$) *In MA, $=_L \;\subseteq\; \simeq_{\mathrm{bis}}$.*

Corollary 4.10 *In MA, relations $=_L$, $\simeq_{\mathrm{bis}}$ and $\sim_{\mathrm{ind}}$ coincide. Further, on $MA_{\mathrm{IF}}^{\mathrm{syn}}$, they also coincide with $\equiv$.*

Model-checking and tautologies. In AL, the construction of characteristic formulas has connections with other problems related to logic, namely the decidability of model-checking (whether $P \models \mathcal{A}$ holds, for any given process P and formula $\mathcal{A}$) and the decidability of tautologies (whether a given formula $\mathcal{A}$ is satisfied by all processes). These problems have been addressed in [CT01], where the undecidability of tautologies is established on a small fragment of AL, a result that entails the undecidability of model-checking.

Using characteristic formulas, we can derive similar, albeit weaker, results the other way around (they are weaker because the undecidability of tautologies is established on a larger language). First, note that (recall that $\mathcal{F}_S$ is the characteristic formula of process S)

$$\text{for all } P, Q, R \in \mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}, \qquad P \models \mathcal{F}_Q \;\wedge\; \Diamond\mathcal{F}_R \qquad \text{iff} \qquad P \;\equiv\; Q \Longrightarrow R\,.$$

Now, since $\Longrightarrow$ is undecidable on $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$ (we will prove it in Section 5), so is model-checking. Moreover, we can reduce the model-checking problem (on $\mathrm{MA}_{\mathrm{IF}}$) to the tautology problem:

$$\text{for all } P \in \mathrm{MA}_{\mathrm{IF}},\ \text{for all } \mathcal{A}, \quad P \models \mathcal{A} \quad \text{iff} \quad \vdash\; \mathcal{F}_P \to \mathcal{A}\,.$$

As a consequence, the tautology problem is undecidable too.

5 (Un)decidability of the logical equivalence

The undecidability of $=_L$ on MA is obtained via an encoding of Turing Machines (TM's) in the subcalculus $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$.

The encoding and its correctness proof are conceptually simple. The proof, however, is long and tedious, due to the complexity of the TM encoding. Our encoding follows the ideas of Cardelli and Gordon's [CG98]. We had however to add or expand some components, because: (1) we do not have the restriction operator, used in [CG98]; (2) we cannot use coarse behavioural equivalences such as testing or barbed equivalence to reason on processes, as customary in process calculus encodings; we are only allowed to use $\equiv$, which is a very strong equivalence (on $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$, $\equiv$ and $=_L$ coincide, Theorem 3.8); therefore, for instance, we cannot algebraically garbage collect deadlocked processes: we have to add into the encoding special processes that explicitly perform garbage collection; (3) we need the simulation of a TM to be (almost) deterministic; to obtain this, we have to add some components that force sequentialisations. We are not aware of correctness results concerning the encoding of [CG98].

A TM is defined by a *ribbon*, a *transition relation* on some set of *states*, *initial* and *accepting states*. A ribbon is a finite sequence of cells, each containing a binary information. In the encoding, a ribbon of length k is represented by a nesting of k ambients named *cell*. Each such ambient has a subambient $d[\mathbf{0}]$, where $d \in \{tt, ff\}$ represents the content of the cell. The Turing machine moves left and right by exercing in *cell* and out *cell* capabilities. After each movement, an ambient representing the head of the machine reads the value of the current cell, rewrites it, and triggers the next movement.

For the correctness, the central result roughly says that if a given Turing machine $\mathcal{M}$ in state S_0 recognises a word w of input and terminates in a state S_t with a word w' on the ribbon, then

$$\mathtt{TM}(w, S_0) \rightsquigarrow^* \mathtt{TM}(w', S_t)\,,$$

where $\mathtt{TM}(v, S)$ represents the encoding of the TM in state S and with v as content of the ribbon, and the relation $\rightsquigarrow^*$ is defined as follows: write $P \rightsquigarrow Q$ if $P \longrightarrow Q$ and for any Q' such that $P \longrightarrow Q'$, either $Q' \not\longrightarrow$ or $Q \equiv Q'$; then $\rightsquigarrow^*$ is the reflexive and transitive closure of $\rightsquigarrow$. Thus $P \rightsquigarrow^* Q$ says that P can reduce to Q and, moreover, the reduction is almost deterministic. (What prevents pure determinism are blocking states that arise in the encoding of if-then-else statements in the TM.)

Moreover, we relate the halting problem of TM's to the existence of certain loops: given a Turing machine $\mathcal{M}$ and an input word w, there are appropriate processes P_0 and P_1 obtained from the encoding of the TM such that

$$\mathcal{M} \text{ halts on input } w \quad \text{iff} \quad P_0 \Longrightarrow P_1 \Longrightarrow P_0. \tag{2}$$

Theorem 5.1 *$=_L$ is an undecidable relation on MA.*

Proof: Consider processes P_0 and P_1 from (2). These processes are in $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$. Using Corollary 4.10, the definition of $\simeq_{\mathrm{bis}}$, and Theorem 3.8, we have:

$$\mathsf{open}\; n.\, P_0 =_L \mathsf{open}\; n.\, P_1 \quad \text{iff} \quad \mathsf{open}\; n.\, P_0 \simeq_{\mathrm{bis}} \mathsf{open}\; n.\, P_1 \quad \text{iff} \quad P_0 \Longrightarrow \simeq_{\mathrm{bis}} P_1 \Longrightarrow \simeq_{\mathrm{bis}} P_0$$

(from Theorem 3.8, $\Longrightarrow \simeq_{\mathrm{bis}}$ is $\Longrightarrow$ on $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$). Then undecidability follows from (2). □

Proposition 5.2 (Decidability of $=_L$ on $\mathbf{MA}_{\mathrm{IF}}$ and $\mathbf{MA}_{\mathrm{IF}}^{\mathrm{syn}}$) *$=_L$ is a decidable relation on MA_{IF} and $MA_{\mathrm{IF}}^{\mathrm{syn}}$.*

Proof: These results are proved using the inductive characterisation of $=_L$ (Corollary 4.10) and the image-finiteness conditions in the definitions of the calculi. For $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$ the result also follows directly from Theorem 3.8. □

The encoding of TM's only uses processes in $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$. The language $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$ is therefore proved to be Turing complete. This might seem in contradiction with the decidability of $=_L$ in $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$. The proof of Theorem 5.1 does not work for $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$ because $\mathsf{open}\; n.\, P_0$ and $\mathsf{open}\; n.\, P_1$ do not belong to this language. Indeed, concerning $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$ we can only derive, from (2), that *reachability* (whether $P \Longrightarrow P'$ holds, for any P, P') is undecidable.

6 Extensions

The syntax of MA in [CG00] also includes communication, i.e., operators $\langle V \rangle$ for the emission of a value, and $(x)P$ for reception. The value V can be a name, a capability, or a path of capabilities (a string of capabilities).

The results we have presented can be extended to MA with communication of names. In the statement of the results, the main difference is that on $\mathrm{MA}_{\mathrm{IF}}^{\mathrm{syn}}$, $=_L$ coincides with $\equiv_{\mathrm{E}}$, the (decidable) relation obtained by adding the eta-equality

$$(x)\big(\langle x \rangle \mid (y)P\big) \;=\; (y)P$$

to the axioms of $\equiv$ (a similar result was known for finite MA [San01]). We believe that also the addition of communication of capabilities is easy to handle. By contrast, we do not know how to handle communication of paths of capabilities, because the sequential depth of processes may then grow under reduction, and many of our lemmas are proved by induction on this measure (cf. Lemma 2.2).

Usually [CG98, CG99], the syntax of MA also has the restriction operator. In [CG01b], Cardelli and Gordon propose an extension of AL with logical connectives to describe restriction. We do not know at present whether our results continue to hold with such an extension. Also, we do not know whether the results hold for an MA calculus with a recursion operator instead of replication, since recursion gives us trees with infinite depth.

References

[Car99] L. Cardelli. Semistructured computations. Proc. 7th Intl. workshop on Data Base Programming Languages (DBPL'99), invited talk (accompanying paper available from the author's web page), 1999.

[CC01] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). In *Proc. of TACS'01*, LNCS. Springer Verlag, 2001.

[CG98] L. Cardelli and A.D. Gordon. Mobile ambients. In Nivat. M., editor, *Proc. FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998.

[CG99] L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proc. 26th POPL*, pages 79–92. ACM Press, 1999.

[CG00] L. Cardelli and A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. 27th POPL*. ACM Press, 2000.

[CG01a] 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.

[CG01b] L. Cardelli and A. Gordon. Logical Properties of Name Restriction. In *Proc. of TLCA'01*, volume 2044 of *LNCS*. Springer Verlag, 2001.

[CT01] W. Charatonik and J-M. Talbot. The Decidability of Model Checking Mobile Ambients. In *Proc. of CSL'01*, LNCS. Springer LNCS, 2001.

[DZ00] S. Dal-Zilio. Structural Congruence for Ambients is Decidable. In *Proc. of ASIAN'00*, volume 1961 of *LNCS*. Springer Verlag, 2000.

[EG99] J. Engelfriet and T. Gelsema. Multisets and structural congruence of the π-calculus with replication. *TCS*, 211(1-2):311–337, 1999.

[GS86] S. Graf and J. Sifakis. A modal characterization of observational congruence on finite terms of CCS. *Information and Control*, 68:125–145, 1986.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[How96] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

[San01] D. Sangiorgi. Extensionality and Intensionality of the Ambient Logic. In *Proc. of 28th POPL'01*, pages 4–17. ACM Press, 2001.

[SI94] B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.

A Logical formulas

We begin with the definition of the formula to capture the set of free names of a process.

$$
\begin{array}{lll}
m = n & \overset{\mathrm{def}}{=} & (n[\top])@m \qquad\qquad \text{(this formula is from [CG00])} \\
\mathsf{flat}\ m & \overset{\mathrm{def}}{=} & \llbracket \mathsf{in}\ m \rrbracket.0\ \vee\ \llbracket \mathsf{out}\ m \rrbracket.0\ \vee\ \llbracket \mathsf{open}\ m \rrbracket.0\ \vee m[0] \\
\mathsf{flat\ cond.}(n) & \overset{\mathrm{def}}{=} & \big(\exists m.\ \neg(m = n)\ \wedge\ \mathsf{flat}\ m\ \big)^{\omega} \\
\mathsf{top\ level\ cond.}(n) & \overset{\mathrm{def}}{=} & \big(\langle \mathsf{in}\ n\rangle.\top\ \ \vee\ \ \langle \mathsf{out}\ n\rangle.\top\ \ \vee\ \ \langle \mathsf{open}\ n\rangle.\top\ \ \vee\ \ n[\top]\ \big)\ \big|\ \top \\
\mathsf{refers1}\ n & \overset{\mathrm{def}}{=} & \forall m.\ \mathsf{flat\ cond.}(n) \blacktriangleright \big(\ \mathsf{flat\ cond.}(n) \blacktriangleright \diamond\ m[\mathsf{top\ level\ cond.}(n)]\ \big)@m \\
\mathsf{refers}\ \{n_1, \ldots, n_k\} & \overset{\mathrm{def}}{=} & \bigwedge_{i=0\ldots k} \mathsf{refers1}\ n_i\ \wedge\ \forall x.\ \mathsf{refers1}\ x\ \rightarrow\ \bigvee_{i=0\ldots k} x = n_i
\end{array}
$$

Formula $\mathsf{refers1}\ n$ detects whether name n occurs in a process, while $\mathsf{top\ level\ cond.}(n)$ detects whether n occurs at top level. In $\mathsf{refers}\ \{n_1, \ldots, n_k\}$, when $k = 0$ the indexed conjunction is replaced by 'true', and the indexed disjunction by 'false'.

We now move to the formulas capturing (ephemeral and replicated) single processes. The following definitions give the details of the construction behind $!\mathcal{A}$ in the main text (using an auxiliary function Rep that makes explicit the dependency from the outermost operator of processes).

$$
\begin{array}{lll}
\mathcal{A}^{\omega} & \overset{\mathrm{def}}{=} & (\mathsf{1Comp} \rightarrow \mathcal{A})^{\forall} \qquad\qquad \mathcal{F}_{\mathsf{cap}.0} \overset{\mathrm{def}}{=} \llbracket \mathsf{cap} \rrbracket.0 \\
\mathsf{Rep}_{\mathsf{in}\ n}(\mathcal{A}) & \overset{\mathrm{def}}{=} & \mathcal{A}^{\omega}\ \ \wedge\ \ \forall m.\ (\neg \mathsf{refers1}\ m)\ \rightarrow \\
 & & (\mathcal{F}_{\mathsf{out}\ n.0})^{\omega} \triangleright \big(\ n[0] \triangleright \Box\diamond\ (\ n[m[\mathcal{A}\ \mid\ \top]]\ )\big)@m \\
\mathsf{Rep}_{\mathsf{out}\ n}(\mathcal{A}) & \overset{\mathrm{def}}{=} & \mathcal{A}^{\omega}\ \ \wedge\ \forall m.\ (\neg \mathsf{refers1}\ m)\ \rightarrow \\
 & & (\mathcal{F}_{\mathsf{in}\ n.0})^{\omega} \triangleright \big(\ n[0] \triangleright \Box\ \diamond(\ m[\mathcal{A}\ \mid\ \top] \mid n[\ ])\big)@m \\
\mathsf{Rep}_{\mathsf{open}\ n}(\mathcal{A}) & \overset{\mathrm{def}}{=} & \mathcal{A}^{\omega}\ \wedge\ (n[0])^{\omega} \triangleright \Box\ (\ \mathcal{A}\ \mid\ \top\ ) \\
\mathsf{Rep}_{n[]}(\mathcal{A}) & \overset{\mathrm{def}}{=} & (n[\mathcal{A}])^{\omega}\ \wedge\ (\mathcal{F}_{\mathsf{open}\ n.0})^{\omega} \triangleright \Box\ (n[\mathcal{A}]\ \mid\ \top\ )
\end{array}
$$

The interpretation of these formulas necessitates some hypotheses concerning the sequentiality and depth degree, defined as follows (we are more refined here than in the main text; compare Lemmas 4.2 and A.3):

Definition A.1 (Sequentiality and depth degree) *The sequentiality (resp. depth) degree of a process is given by a function* ds *(resp.* dd*), from MA processes into natural numbers, inductively defined by:*

$$
\begin{array}{ll}
\mathsf{ds}(\mathbf{0}) \overset{\mathrm{def}}{=} 0 & \mathsf{dd}(\mathbf{0}) \overset{\mathrm{def}}{=} \mathsf{dd}(\mathit{cap}.P) \overset{\mathrm{def}}{=} 0 \\
\mathsf{ds}(\mathit{cap}.P) \overset{\mathrm{def}}{=} 1 + \mathsf{ds}(P) & \mathsf{dd}(n[P]) \overset{\mathrm{def}}{=} \mathsf{dd}(P) + 1 \\
\mathsf{ds}(n[P]) \overset{\mathrm{def}}{=} \mathsf{ds}(!P) \overset{\mathrm{def}}{=} \mathsf{ds}(P) & \mathsf{dd}((!)P_1 \mid \ldots \mid (!)P_r) \overset{\mathrm{def}}{=} \max_{1 \le i \le r} \mathsf{dd}(P_i) \\
\mathsf{ds}(P \mid Q) \overset{\mathrm{def}}{=} \max\big(\mathsf{ds}(P), \mathsf{ds}(Q)\big) &
\end{array}
$$

Definition A.2 (Sequential and depth selectivity) *We say that a formula is* sequentially *(resp.* depth*)* selective *if all processes satisfying it have the same sequentiality (resp. depth) degree.*

For any cap (resp. n), we also say that $\mathcal{A}$ is *cap-expressive* (resp. *n-expressive*) if all terms satisfying it are of the form $\mathsf{cap}.P$ (resp. $n[P]$).

Lemma A.3 1. *Given* cap*, and a sequentially selective and* cap*-expressive formula* $\mathcal{A}$*, define* $!\mathcal{A} \overset{\mathrm{def}}{=} \mathit{Rep}_{\mathit{cap}}(\mathcal{A})$*. Then property (1) of Section 4 holds, replacing each* P_i *with* $\mathsf{cap}.P_i$*.*

2. *Similarly, for any name* n *and depth selective and* n*-expressive formula* $\mathcal{A}$*, define* $!\mathcal{A} \overset{\mathrm{def}}{=} \mathit{Rep}_{n[]}(\mathcal{A})$*. Then property (1) of Section 4 holds, replacing each* P_i *with* $n[P_i]$*.*

To construct a characteristic formula $\mathcal{F}_P$ of a process P, we may suppose (up to $\equiv$) that replication only appears above single terms. We then reason by structural induction as follows:

$$\begin{array}{lcl@{\qquad}lcl}
\mathcal{F}_{\mathbf{0}} & \stackrel{\text{def}}{=} & 0 & \mathcal{F}_{\mathsf{cap}.P} & \stackrel{\text{def}}{=} & \langle\mathsf{cap}\rangle.\,\mathcal{F}_P \;\wedge\; [\![\mathsf{cap}]\!].\,\bigvee_{\{P',\ P\Longrightarrow P'\}_{/\simeq_{\text{bis}}}} \mathcal{F}_{P'} \\
\mathcal{F}_{n[P]} & \stackrel{\text{def}}{=} & n[\mathcal{F}_P] & \mathcal{F}_{!n[P]} & \stackrel{\text{def}}{=} & \mathsf{Rep}_{n[]}(\mathcal{F}_P) \\
\mathcal{F}_{P|Q} & \stackrel{\text{def}}{=} & \mathcal{F}_P \;\mid\; \mathcal{F}_Q & \mathcal{F}_{!\mathsf{cap}.P} & \stackrel{\text{def}}{=} & \mathsf{Rep}_{\mathsf{cap}}(\mathcal{F}_{\mathsf{cap}.P})
\end{array}$$

B Turing Machine implementation

The Turing machine encoding is defined using (possibly parametrised) contexts in which the hole is written $\{\!|\;|\!\}$. Some parametrised definitions shall be written $\mathtt{foo}(p); P$ (here `foo` depends on parameters p and P), to emphasise a form of sequentiality.

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 as $w^{(1)} \dots w^{(r)}$ to refer to w's digits.

$$\begin{array}{lcl}
\textbf{Cells and Words} & & \\
\mathtt{cell}(d)\{\!|\;|\!\} & \stackrel{\text{def}}{=} & cell[d[\mathbf{0}] \;\mid\; !\mathsf{open}\ wo \;\mid\; \{\!|\;|\!\}\ ] \\
\mathtt{word}(w)\{\!|\;|\!\} & \stackrel{\text{def}}{=} & \mathtt{cell}(w^{(1)})\{\!|\ \mathtt{cell}(w^{(2)})\{\!|\ \dots \mathtt{cell}(w^{(r)})\{\!|\;|\!\} \dots\ |\!\}\ |\!\} \quad (w = w^{(1)}w^{(2)}\dots w^{(r)}) \\
\textbf{Ribbon Extensor} & & \\
\mathtt{deadextcode} & \stackrel{\text{def}}{=} & !\mathsf{open}\ coin.\,\mathsf{open}\ newcell.\,\mathsf{in}\ cell.\,coin[\mathbf{0}] \;\mid\; !newcell[\mathtt{cell}(f\!f)\{\!|\ \mathsf{out}\ ext\ |\!\}] \\
\mathtt{sendstart} & \stackrel{\text{def}}{=} & msg[\mathsf{out}\ ext \;\mid\; !\mathsf{out}\ cell \;\mid\; \mathsf{out}\ ribbon\_left.\,start[\mathsf{in}\ TM]] \\
\mathtt{ExtensorFrozen} & \stackrel{\text{def}}{=} & ext[\mathtt{deadextcode} \;\mid\; \mathsf{open}\ coin.\,\mathtt{sendstart}] \\
\mathtt{ExtensorAlive} & \stackrel{\text{def}}{=} & ext[coin[\mathbf{0}] \;\mid\; \mathtt{deadextcode} \;\mid\; \mathsf{open}\ coin.\,\mathtt{sendstart}] \\
\mathtt{ExtensorDead} & \stackrel{\text{def}}{=} & ext[\mathtt{deadextcode}] \\
\textbf{Ribbons} & & \\
\mathtt{cleaninst} & \stackrel{\text{def}}{=} & \mathsf{open}\ cleaner.\,\mathsf{open}\ runclean \;\mid\; runclean[\mathtt{deadcleancode}] \\
\mathtt{deadcleancode} & \stackrel{\text{def}}{=} & !\mathsf{open}\ f\!f \;\mid\; !\mathsf{open}\ tt \;\mid\; !\mathsf{open}\ cell \;\mid\; !\mathsf{open}\ wo \\
\mathtt{FrozenRibbon}(w) & \stackrel{\text{def}}{=} & ribbon\_left[\mathtt{cleaninst} \;\mid\; \mathtt{word}(w)\{\!|\ \mathtt{ExtensorFrozen}\ |\!\}] \\
\mathtt{GrowingRibbon}(w) & \stackrel{\text{def}}{=} & ribbon\_left[\mathtt{cleaninst} \;\mid\; \mathtt{word}(w)\{\!|\ \mathtt{ExtensorAlive}\ |\!\}] \\
\mathtt{WorkRibbon}(w_1, w_2)\{\!|\;|\!\} & \stackrel{\text{def}}{=} & ribbon\_left[\mathtt{cleaninst}] \;\mid\; \mathtt{word}(w_1)\{\!|\ \{\!|\;|\!\} \;\mid\; \mathtt{word}(w_2)\{\!|\ \mathtt{ExtensorDead}\ |\!\}\ |\!\} \\
\mathtt{OldRibbon} & \stackrel{\text{def}}{=} & 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 below).

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 quadruple $(\mathcal{Q}, q_{start}, q_A, \delta)$ where $\mathcal{Q}$ is a set of states, q_{start} and q_A are the initial and accepting states, and $\delta : \mathcal{Q} \times \{f\!f, tt\} \;\rightarrow\; \mathcal{Q} \times \{f\!f, tt\} \times \{\leftarrow, \downarrow, \rightarrow\}$ is the evolution function.

Turing Machine Transition

$$
\begin{aligned}
&\mathtt{clear}(d); P &&\stackrel{\text{def}}{=} && wo[\ \mathsf{out}\ head.\ \mathsf{open}\ d.\ cl\_ack[\mathsf{in}\ head]\ ] \ \mid\ \mathsf{open}\ cl\_ack.\ P\\
&\mathtt{write}(d); P &&\stackrel{\text{def}}{=} && wo[\ \mathsf{out}\ head.\ d[\mathbf{0}] \mid wr\_ack[\mathsf{in}\ head]\ ] \ \mid\ \mathsf{open}\ wr\_ack.\ P\\
&\mathtt{become}(mo); P &&\stackrel{\text{def}}{=} && mo[\mathsf{out}\ head.\ \mathsf{open}\ head.\ P] \mid \mathsf{in}\ mo\\
&\mathtt{domove}(mv); P &&\stackrel{\text{def}}{=} && \begin{cases} \mathsf{in}\ cell.\ P & \text{if } mv = \leftarrow\\ P & \text{if } mv = \downarrow\\ \mathsf{out}\ cell.\ P & \text{if } mv = \rightarrow \end{cases}\\
&\mathtt{transcode}(d_r, q_w, d_w, mv) &&\stackrel{\text{def}}{=} && \mathtt{clear}(d_r); \mathtt{write}(d_w); \mathtt{become}(mo); \mathsf{in}\ TM.\ \mathtt{domove}(mv); \mathsf{open}\ q_w
\end{aligned}
$$

State

$$
\begin{aligned}
&\mathit{ff} \to P\ +\ tt \to Q &&\stackrel{\text{def}}{=} && coin[\mathsf{in}\ \mathit{ff}.\ \mathsf{out}\ \mathit{ff}.\ P] \mid coin[\mathsf{in}\ tt.\ \mathsf{out}\ tt.\ Q] \mid \mathsf{open}\ coin\\
&\mathsf{code}(q) &&\stackrel{\text{def}}{=} && !q[head[\ \mathsf{out}\ TM.\ (\ \mathit{ff} \to \mathtt{transcode}(\mathit{ff}, d_{\mathit{ff}}, q_{\mathit{ff}}, mv_{\mathit{ff}})\\
& && && \qquad\qquad\qquad + tt \to \mathtt{transcode}(tt, d_{tt}, q_{tt}, mv_{tt}))\ ]]\\
& && && \mid\ !coin[\mathsf{in}\ \mathit{ff}.\ \mathsf{out}\ \mathit{ff}.\ \mathtt{transcode}(\mathit{ff}, d_{\mathit{ff}}, q_{\mathit{ff}}, mv_{\mathit{ff}})]\\
& && && \mid\ !coin[\mathsf{in}\ tt.\ \mathsf{out}\ tt.\ \mathtt{transcode}(tt, d_{tt}, q_{tt}, mv_{tt})]\\
&\mathsf{code}(q_A) &&\stackrel{\text{def}}{=} && !q_A[\ get\_out[\mathbf{0}]\ ]
\end{aligned}
$$

Turing Machine Behaviour after Recognition

$$
\begin{aligned}
&\mathtt{getout} \stackrel{\text{def}}{=}\\
&\quad !\mathsf{open}\ get\_out.\ \mathsf{out}\ cell.\ get\_out[\mathbf{0}]\\
&\mid\ !\mathsf{open}\ get\_out.\ \mathsf{out}\ ribbon\_left.\ (\quad cleaner[\mathsf{out}\ TM.\ \mathsf{in}\ ribbon\_left]\\
&\qquad\qquad\qquad \mid\ coin[\mathsf{out}\ TM.\ \mathsf{in}\ ribbon\_left.\ \mathsf{in}\ cell^{\mathsf{length}(w)}.\ \mathsf{in}\ ext]\\
&\qquad\qquad\qquad \mid\ \mathsf{open}\ start.\ \mathsf{in}\ ribbon\_left.\ \mathsf{in}\ cell.\ \mathsf{open}\ q_{start}\ \ )
\end{aligned}
$$

The encoding of a Turing Machine is based on an ambient named TM, containing a persistent code named `tmsoup`:

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

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

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

Once the computation has started, the Turing machine in state q is represented by the process

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

Note as well that this encoding is parametric over a word w, whose length (denoted $\mathsf{length}(w)$) is used in the definition of `getout`. This is however no harm for our undecidability result.

Proposition B.1 *Given a Turing machine $\mathcal{M}$, for any word w and $k \in \mathbb{N}$, the Turing machine $\mathcal{M}$ recognises the word w without using more than $k + |w|$ cells of the ribbon iff there exist two words w_1 and w_2 such that*

$$\mathtt{WorkRibbon}(\epsilon, w.\mathit{ff}^k)\{\!\!|\ \mathtt{TM}(q_{start})\ |\!\!\} \leadsto^* \mathtt{WorkRibbon}(w_1, w_2)\{\!\!|\ \mathtt{TM}(q_A)\ |\!\!\}.$$

Lemma B.2 *Given a Turing machine $\mathcal{M}$ and a word w, define the following processes, obtained from the encoding of $\mathcal{M}$:*

$$
\begin{aligned}
&Q \stackrel{\text{def}}{=}\ !\mathtt{FrozenRibbon}(w) \mid\ !\mathtt{OldRibbon} \mid\ !\mathit{open}\ msg \mid\ !\mathit{out}\ cell \mid \mathtt{TMStart}\,,\\
&P_0 \stackrel{\text{def}}{=} Q \mid \mathtt{GrowingRibbon}(w)\,, \qquad P_1 \stackrel{\text{def}}{=} Q \mid \mathtt{GrowingRibbon}(w.\mathit{ff})\,.
\end{aligned}
$$

It holds that $P_0 \implies P_1$. Conversely, $P_1 \implies P_0$ if and only if the word w is recognised by $\mathcal{M}$.