Extensionality and Intensionality of the Ambient Logics *

Davide Sangiorgi
INRIA Sophia-Antipolis, France
davide.sangiorgi@sophia.inria.fr

ABSTRACT

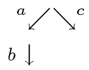
The *ambient logic* has been proposed for expressing properties of process mobility in the calculus of Mobile Ambients (MA), and as a basis for query languages on semistructured data.

To understand the extensionality and the intensionality of the logic, the equivalence on MA processes induced by the logic ($=_{\mathrm{L}}$) is compared with the standard MA behavioural equivalence and with structural congruence (an intensional equivalence, used as an auxiliary relation in the definition of satisfaction of the logic). The main contributions include a co-inductive characterisation of $=_{\mathrm{L}}$ as a form of labelled bisimilarity, and axiomatisations of $=_{\mathrm{L}}$ on the synchronous and asynchronous (finite) calculus.

The study shows that, surprisingly, the logic allows us to observe the internal structure of the processes at a very fine-grained detail, much in the same way as structural congruence does. A spin-off of the study is a better understanding of behavioural equivalence in Ambient-like calculi. For instance, behavioural equivalence is shown to be insensitive to *stuttering* phenomena originated by processes that may repeatedly enter and exit an ambient.

1. INTRODUCTION

The purpose of the calculus of Mobile Ambients, MA [5, 7], is to study process mobility, where mobility is thought of as modifications of spatial configurations over time. Locations, called ambients, are named, and may be nested. A spatial configuration can therefore be represented as an edge-labelled tree in which the subtrees are the subambients and the labels of the edges are the ambient names. For instance, the MA process $a[\,b[\,\mathbf{0}\,]\,] \mid c[\,\mathbf{0}\,]$, formed by an empty ambient c in parallel with an ambient a containing an empty subambient b, has the following tree representation (drawing only the edges):



The trees are unordered: the order of the children of a node is not important.

An ambient may contain, besides other ambients, also local processes, which determine the ambient movements and the communications. (In the tree representation, these local processes can be thought of as the content of the nodes of a tree.) 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. Communication is exchange of messages within an ambient; an example is

$$a[\,\{M\} \mid (x)\,P \mid b[\,Q\,]\,] \longrightarrow a[\,P\{M/x\} \mid b[\,Q\,]\,]$$

in which the message with content M is consumed by the abstraction $(x)\,P$.

To express properties of mobility in MA, Cardelli and Gordon have introduced a modal logic [8]. This is, intuitively, a logic of dynamically-evolving unordered edge-labelled trees. Indeed, the actual definition of satisfaction of the formulas is given on MA processes quotiented by a relation of *structural congruence*, $\equiv$, that equates processes with the same tree representation. (Relation $\equiv$ follows the idea of the Chemical Abstract Machine [2] and Milner's structural congruence for the π-calculus [13].)

The logic has a connective that talks about *time*, that is, how processes can evolve: the formula $\diamond\, A$ is satisfied by those processes with a future in which 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 $n[\,A\,]$ is satisfied by ambients named n whose content satisfies A (read on trees: $n[\,A\,]$ is satisfied by the trees whose root has just a single edge n leading to a subtree that satisfies A); the formula $A_1 \mid A_2$ is satisfied by the processes that can be decomposed into parallel components P_1 and P_2 where each P_i satisfies A_i (read on trees: $A_1 \mid A_2$

*To appear in Proc. 28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages London, January 17-19, 2001.

is satisfied by the trees that are the juxtaposition of two trees that respectively satisfy the formulas A_1 and A_2); the formula $\mathbf{0}$ is satisfied by the terminated process $\mathbf{0}$ (on trees: $\mathbf{0}$ is satisfied by the tree consisting of just the root node).

The ambient logic has also been advocated as a foundation of query languages for semistructured data [3, 4]. 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 the standard models of semistructured data.

In this paper we study the extensionality and the intensionality of the ambient logic. We discuss below the motivations for this topic, and what makes it technically challenging.

A logic, defined on the terms of a calculus or of a model, is *extensional* if it can only separate terms that have different behaviours: processes with the same behaviours satisfy the same sets of formulas. (A stronger notion of extensionality is sometimes preferred, whereby the logic separates *all* and only the terms with different behaviours.) The logic is *intensional* if it can separate terms on the basis of their internal structure, even though their behaviours are the same. In a process calculus, the notions of 'behaviour' and 'behavioural equivalence' are best formalised operationally; for instance, by means of a reduction relation and a contextually-defined equivalence such as may testing [9] or barbed congruence [14].

'Extensionality and intensionality' is therefore a fundamental question for a logic: it tells us whether the logic fits the intended semantics of the calculus or model on which the logic is defined. The question boils down to understanding the equivalence on terms induced by the logic, whereby two terms are equated if they satisfy the same sets of formulas. The topic is well-understood in standard modal logics. A well-known example, in CCS, is the characterisation of bisimilarity in terms of the Hennessy-Milner logic [10].

The ambient logic has however striking differences with standard modal logics. First, standard modal logics do not talk about space. Secondly, they have more precise temporal connectives. The only temporal connective of the ambient logic talks about the 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 the temporal modality $\langle \mu \rangle. A$ is satisfied by the processes that can perform the action μ thus becoming a process that satisfies A. The action μ can be a reduction, but also an input or an output. The lack of temporal connectives in the ambient logic is particularly significant because in MA interaction between a process and its environment can take several forms, originated by the communication and the movement primitives. (There are 9 such forms; they appear as labels of transitions in a purely SOS semantics of MA [6, 12].)

We are therefore led to comparing three relations on MA processes: the equivalence induced by the ambient logic, $=_{\mathrm{L}}$; behavioural equivalence, $\approx$ (as the logic is branching-time, $\approx$ is barbed congruence, which is based on bisimulation); and structural congruence, $\equiv$. The comparison with $\equiv$ is important because $\equiv$ is intensional and because it is used as an auxiliary relation in the definition of satisfaction in the logic. To begin with, we restrict ourselves to the finite processes, i.e., the processes without recursion or replication. One of the reasons why finite processes are interesting is that on them we can hope to axiomatise the differences among the equivalences. Afterwards, we discuss the addition of recursion.

Except for the inclusion $\equiv \subseteq =_{\mathrm{L}}$ (which holds by definition of satisfaction), the relationship of $=_{\mathrm{L}}$ with $\equiv$ and $\approx$ is far from obvious. It is not obvious whether $=_{\mathrm{L}} \subseteq \approx$ should hold because the potential interactions of a process with the environment (communications and movements) are observable in the behavioural equivalence but, as mentioned above, the logic does not have primitive temporal connectives for them. The opposite inclusion is not obvious either, because the logic allows us to intensionally look into the parallel structure of processes. Finally, it is not obvious that $=_{\mathrm{L}} \subseteq \equiv$ should hold because $\equiv$ is a very strong relation – a few axioms are the only difference with syntactic identity.

We summarise our main contributions. We show that $=_{\mathrm{L}}$ is strictly included in $\approx$ and that, on the finite processes, the only difference between $=_{\mathrm{L}}$ and $\equiv$ is an axiom similar to the eta rule of the λ-calculus. In MA, communication is asynchronous. We also show that $=_{\mathrm{L}}$ is precisely $\equiv$ in the (finite) *synchronous* calculus. To obtain the results we go through two co-inductive characterisations of $=_{\mathrm{L}}$, as forms of labelled bisimilarity. The reason for having *two* such characterisations is technical, and has to do with the well-known problem of proving that bisimilarity is a congruence in higher-order process calculi. (MA is higher-order because terms may move during a computation.) The hardest part of the co-inductive characterisations is, however, the definition of the characteristic formulas for the messages, that is, the formulas that separate the messages with a given content from the other processes.

The study shows that, surprisingly, the logic allows us to observe the internal structure of the processes at a very fine-grained detail, much in the same way as structural congruence does. A spin-off of the study is a better understanding of behavioural equivalence in MA-like calculi, something little explored so far. For instance, we have discovered that behavioural equivalence is insensitive to *stuttering* phenomena, originated by processes that may repeatedly enter and exit an ambient.

The characterisations in terms of $\equiv$ plus eta and in terms of $\equiv$ give us axiomatisations of $=_{\mathrm{L}}$ on asynchronous and synchronous finite MA. We are not aware of other axiomatisations of semantic equivalences (defined by operational, denotational, logical, or other means) in higher-order process calculi. As counterexamples to the inclusion $\approx \subseteq =_{\mathrm{L}}$, we have found three axiom schemata. We do not know whether they are complete, that is, they exactly describe the difference between the two relations on finite MA.

2. BACKGROUND

This section collects the necessary background: the MA calculus, the ambient logic. Sections 3 to 6 will present our contributions.

2.1 Finite Ambients

We recall here the syntax of finite MA, briefly MA, from [5]. The adjective 'finite' indicates the omission of operators for representing infinite terms, such as recursion or replication. Furthermore, only names, not capabilities, can be communicated (this allows us to work in an untyped calculus). As in [8, 3, 4], the calculus has no restriction operators for creating new names. It is not clear how to best handle restriction in the logic. The restriction-free calculus, however, has a more direct correspondence with edge-labelled trees and semistructured data.

Table 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 that of variables are infinite. A *closed* process has no free variables. The expressions $\mathtt{in}\,\eta$, $\mathtt{out}\,\eta$, and $\mathtt{open}\,\eta$ are the *capabilities*. Messages and abstractions are the *input/output* (I/O) primitives.

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

$$\begin{aligned} P \mid \mathbf{0} &\equiv P \\ P \mid Q &\equiv Q \mid P \\ P \mid (Q \mid R) &\equiv (P \mid Q) \mid R. \end{aligned}$$

The rules for the reduction relation, $\longrightarrow$, are reported in Table 2. The reflexive and transitive closure of $\longrightarrow$ is written $\Longrightarrow$. To accommodate the extensions of MA of Section 6, where also capabilities can be communicated, in Table 2 (as well as in Definitions 3.1 and 3.2 below) we write M for the content of a generic message.

Behavioural equivalence is defined on top of reduction and of 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 2.1 (BARBED CONGRUENCE, [14, 12]). Barbed bisimulation *is the largest symmetric relation* $\dot{\approx}$ *on closed processes such that* $P \ \dot{\approx}\ Q$ *implies:*

1. *whenever* $P \Longrightarrow P'$ *then there exists* Q' *such that* $Q \Longrightarrow Q'$ *and* $P' \ \dot{\approx}\ Q'$*;*

2. *for each name* n*,* $P \Downarrow_n$ *iff* $Q \Downarrow_n$.

Two processes P *and* Q *are* barbed congruent*, written* $P \approx Q$*, if* $C[P] \ \dot{\approx}\ C[Q]$ *for all closing contexts* C.

2.2 The logic

This is the set of the formulas of the logic:

$$\begin{array}{rcll} A, B & ::= & \mathbf{T} & \textit{(true)} \\ & \mid & \neg A & \textit{(negation)} \\ & \mid & A \vee B & \textit{(disjunction)} \\ & \mid & \mathbf{0} & \textit{(void)} \\ & \mid & \eta[\,A\,] & \textit{(location)} \\ & \mid & A_1 \mid A_2 & \textit{(composition)} \\ & \mid & \forall\, x\,.\,A & \textit{(universal name quantification)} \\ & \mid & \Diamond\, A & \textit{(sometime modality)} \\ & \mid & A@\eta & \textit{(location adjunct)} \\ & \mid & A \rhd B & \textit{(linear implication)} \end{array}$$

The propositional connectives, $\mathbf{T}, \neg A, A \vee B$, and the universal quantification on names, $\forall\, x\,.\,A$, have the standard logical interpretation. The temporal connective, $\Diamond\, A$, and the spatial connectives, $\mathbf{0}$, $A \mid B$, and $\eta[\,A\,]$, have been briefly discussed in the Introduction. $A \rhd B$ and $A@\eta$ are the logical adjuncts of $A \mid B$ and $\eta[\,A\,]$, in the sense of being, roughly, their inverse. A formula is *closed* if it does not contain free variables.

DEFINITION 2.2. *The* satisfaction *relation is defined on closed processes and closed formulas:*

$$\begin{array}{lcl} P \models \mathbf{T} & \stackrel{\text{def}}{=} & \textit{always true} \\ P \models \neg A & \stackrel{\text{def}}{=} & \textit{not } P \models A \\ P \models A \vee B & \stackrel{\text{def}}{=} & P \models A \textit{ or } P \models B \\ P \models \mathbf{0} & \stackrel{\text{def}}{=} & P \equiv \mathbf{0} \\ P \models n[\,A\,] & \stackrel{\text{def}}{=} & \textit{there is } P' \textit{ such that} \\ & & P \equiv n[\,P'\,] \textit{ and } P' \models A \\ P \models A_1 \mid A_2 & \stackrel{\text{def}}{=} & \textit{there are } P_1, P_2 \textit{ such that} \\ & & P \equiv P_1 \mid P_2 \textit{ and } P_i \models A_i,\ i = 1, 2 \\ P \models \forall\, x\,.\,A & \stackrel{\text{def}}{=} & \textit{for all } n,\ P \models A\{^n\!/\!_x\} \\ P \models \Diamond\, A & \stackrel{\text{def}}{=} & \textit{there is } P' \textit{ such that} \\ & & P \Longrightarrow P' \textit{ and } P' \models A \\ P \models A@n & \stackrel{\text{def}}{=} & n[\,P\,] \models A \\ P \models A \rhd B & \stackrel{\text{def}}{=} & \textit{for all } R \textit{ such that } R \models A, \\ & & \textit{we have } R \mid P \models B \end{array}$$

DEFINITION 2.3 (PROCESS LOGICAL EQUIVALENCE). *For closed processes* P, Q*, we write* $P =_{\mathrm{L}} Q$ *if for all closed formulas* A *it holds that* $P \models A$ *iff* $Q \models A$.

We shall use the dual of some modalities. The dual of the linear implication ($A \blacktriangleright B$), the dual of the sometime modality ($\Box A$), and the standard dual of the universal quantification, ($\exists\, x\,.\,A$) and of the $\vee$-connective ($A \wedge B$):

$$\begin{aligned} A \blacktriangleright B &\stackrel{\text{def}}{=} \neg\,(A \rhd \neg\, B) \\ \Box A &\stackrel{\text{def}}{=} \neg \Diamond \neg\, A \\ \exists\, x\,.\,A &\stackrel{\text{def}}{=} \neg \forall\, x\,.\,\neg\, A \\ A \wedge B &\stackrel{\text{def}}{=} \neg\,(\neg\, A \vee \neg\, B) \end{aligned}$$

			Expressions					Processes
M, N	$::=$	x	*(variable)*		P, Q, R	$::=$	$\mathbf{0}$	*(nil)*
	$\mid$	n	*(name)*			$\mid$	$P \mid Q$	*(parallel)*
	$\mid$	$\mathtt{in}\,\eta$	*(enter)*			$\mid$	$M.\,P$	*(prefixing)*
	$\mid$	$\mathtt{out}\,\eta$	*(exit)*			$\mid$	$\eta[\,P\,]$	*(ambient)*
	$\mid$	$\mathtt{open}\,\eta$	*(open)*			$\mid$	$\{\eta\}$	*(message)*
						$\mid$	$(x)\,P$	*(abstraction)*

Table 1: The syntax of finite MA

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

$$\frac{}{m[\,n[\,\mathtt{out}\,m.\,P_1 \mid P_2\,] \mid Q\,] \longrightarrow n[\,P_1 \mid P_2\,] \mid m[\,Q\,]}\ \texttt{Red-Out} \qquad \frac{}{\{M\} \mid (x)\,P \longrightarrow P\{^{M}\!/\!_{x}\}}\ \texttt{Red-Com}$$

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

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

Table 2: The rules for reduction

Thus $P \models A \blacktriangleright B$ iff there is Q with $Q \models A$ and $P \mid Q \models B$, and $P \models \Box A$ iff $P' \models A$ for all P' such that $P \Longrightarrow P'$. We give $\vee$ and $\wedge$ the least syntactic precedence, thus $A \rhd B \wedge C$ reads $(A \rhd B) \wedge C$, and $A \rhd (\Diamond\, B \wedge \Diamond\, C)$ reads $A \rhd ((\Diamond\, B) \wedge (\Diamond\, C))$.

Lemma 2.4. *If $P \equiv Q$ and $P \models A$ then also $Q \models A$.*

The ambient logic in [8] has also a *somewhere* connective that holds of a process that contains, at some arbitrary level of nesting of ambients, an ambient whose content satisfies A. We have omitted this connective here: its addition would not change the results in the paper. (Formally, this holds because the somewhere connective does not allow us to separate processes that are intensionally bisimilar, Definition 3.2.)

3. INTENSIONAL BISIMILARITY

We look for a co-inductive characterisation of $=_{\mathrm{L}}$, as a form of labelled bisimilarity. In this section we present the bisimilarity and prove that it is a congruence. In the next section we prove that it coincides with $=_{\mathrm{L}}$. First we have to define labelled transitions. We also need relations of *stuttering*.

Again, to accommodate the extensions of MA of Section 6, in the two definitions below we use M as label of transitions also in places where, with the present the syntax for MA, M can only be a name.

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

- *$P \xrightarrow{M} P'$, where M is a capability, if $P \equiv M.\,P_1 \mid P_2$ and $P' = P_1 \mid P_2$.*
- *$P \xrightarrow{!M} P'$ if $P \equiv \{M\} \mid P'$.*
- *$P \xrightarrow{?M} P'$ if $P \equiv (x)\,P_1 \mid P_2$ and $P' \equiv P_1\{^{M}\!/\!_{x}\} \mid P_2$. All transitions $P \xrightarrow{?M} P'$ are supposed to be well-typed, that is, $\{M\} \mid P$ should be a legal closed process (thus, with the syntax of Table 1, M can only be a name).*
- *$P \stackrel{\mu}{\Longrightarrow} P'$, where μ is one of the above labels, if $P \Longrightarrow \xrightarrow{\mu} \Longrightarrow P'$ (where $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$ 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 $1 \leq r < i$.*

We show the definition of bisimilarity, then we comment it.

Definition 3.2. Intensional bisimilarity *is the largest symmetric relation $\simeq$ on closed processes such that $P \simeq Q$ 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 $P_i \simeq Q_i$, 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* $P' \simeq Q'$.

4. *If* $P \xrightarrow{\mathtt{in}\,n} P'$ *then there is* Q' *such that* $Q \stackrel{\mathtt{in}\,n}{\Longrightarrow} \xRightarrow{(\mathtt{out}\,n,\mathtt{in}\,n)^*} Q'$ *and* $P' \simeq Q'$.

5. *If* $P \xrightarrow{\mathtt{out}\,n} P'$ *then there is* Q' *such that* $Q \stackrel{\mathtt{out}\,n}{\Longrightarrow} \xRightarrow{(\mathtt{in}\,n,\mathtt{out}\,n)^*} Q'$ *and* $P' \simeq Q'$.

6. *If* $P \xrightarrow{\mathtt{open}\,n} P'$ *then there is* Q' *such that* $Q \stackrel{\mathtt{open}\,n}{\Longrightarrow} Q'$ *and* $P' \simeq Q'$.

7. *If* $P \xrightarrow{!M} P'$ *then there is* Q' *such that* $Q \stackrel{!M}{\Longrightarrow} Q'$ *and* $P' \simeq Q'$.

8. *If* $P \xrightarrow{?M} P'$ *then there is* Q' *such that* $Q \mid \{M\} \Longrightarrow Q'$ *and* $P' \simeq Q'$.

9. *If* $P \equiv n[\,P'\,]$ *then there is* Q' *such that* $Q \equiv n[\,Q'\,]$ *and* $P' \simeq Q'$.

The definition of $\simeq$ has (at least) two intensional clauses, namely (1) and (2), which allow us to observe parallel compositions and the terminated process. These clauses correspond to the intensional connectives '$|$' and '$\mathbf{0}$' 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.

The most peculiar aspect of the definition of $\simeq$ is the use of the stuttering relations. Although they could be avoided on the finite processes, they cannot in full calculus, as we discuss in Section 6. To convey some intuition for the need of stuttering, here is a simple example using a sum operator *à la* CCS; the processes

$$P \stackrel{\text{def}}{=} \mathtt{in}\,n.\,\mathtt{out}\,n.\,\mathtt{in}\,n.\,P'$$
$$Q \stackrel{\text{def}}{=} P + \mathtt{in}\,n.\,P'$$

are extensionally equal ($P \approx Q$), and they are also equal in the logic ($P =_{\mathrm{L}} Q$). But they would not be intensionally bisimilar ($P \simeq Q$) without the stuttering relations. To see why the extra summand of Q does not affect its behaviour, consider a reduction produced by this summand:

$$m[\,Q \mid R\,] \mid n[\,S\,] \longrightarrow n[\,S \mid m[\,P' \mid R\,]\,]$$

Process P can match it using three reductions:

$$\begin{aligned} m[\,P \mid R\,] \mid n[\,S\,] &\longrightarrow n[\,S \mid m[\,\mathtt{out}\,n.\,\mathtt{in}\,n.\,P' \mid R\,]\,] \\ &\longrightarrow m[\,\mathtt{in}\,n.\,P' \mid R\,] \mid n[\,S\,] \\ &\longrightarrow n[\,S \mid m[\,P' \mid R\,]\,] \end{aligned}$$

By contrast, stuttering does not show up in the Safe Ambients [12], where movements are achieved by means of synchronisations between a capability and a *co-capability.*

It is straightforward to prove that $\simeq$ is an equivalence relation. However, it is not obvious that $\simeq$ should be preserved by all operators of the calculus, due to the fact that $\simeq$ is, intrinsically, higher-order. Formally, $\simeq$ is not higher-order, in that the labels of actions do not contain terms. 'Higher-orderness' is however hidden in clause (3), for a reduction may involve movement of terms (for instance, if the reduction uses rules `Red-In` and `Red-Out`). This, as usual in higher-order forms of bisimilarity, complicates the proof that bisimilarity is preserved by parallel composition.

Our proof of congruence makes use of a second bisimilarity, $\simeq'$, that, by construction, is preserved by all operators of the calculus. By contrast, it is not obvious that $\simeq'$ is preserved under reductions and, consequently, that it is transitive. This proof method, which exploits an auxiliary relation that it 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 [11]. 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 may not be copied as a consequence of a movement. We may say that MA is a *linear* higher-order calculus.

The differences between the definitions of $\simeq$ and $\simeq'$ are the following. First, the structural congruence and the labelled transition in the hypothesis of each clause of $\simeq$ is replaced by syntactic indentity; for instance, clause (1) becomes "if $P = P_1 \mid P_2$ then ...", and clause (4) becomes "if $P = \mathtt{in}\,n.\,P'$ then ...". Secondly, clause (3) is removed. This immediately gives us a relation with the desired property:

LEMMA 3.3. *If* $P \simeq' Q$ *then* $C[P] \simeq' C[Q]$, *for all contexts* C.

To prove that $\simeq$ and $\simeq'$ coincide, the main result we need is that $\simeq'$ is preserved under reductions:

LEMMA 3.4. *Suppose* $P \simeq' Q$ *and* $P \longrightarrow P'$. *Then there is* Q' *such that* $Q \Longrightarrow Q'$ *and* $P' \simeq' Q'$.

PROOF. First we prove a special form of transitivity for $\simeq'$: if $P \equiv P' \simeq' P''$, then $P \simeq' P''$. This allows us to derive $\equiv \subseteq \simeq'$. We then prove the lemma by induction on the depth of the derivation of $P \longrightarrow P'$. $\square$

THEOREM 3.5. *Relations* $\simeq$ *and* $\simeq'$ *coincide, and are congruence relations.*

4. CO-INDUCTIVE CHARACTERISATION

We show that $\simeq$ characterises $=_{\mathrm{L}}$.

THEOREM 4.1. *If* $P \simeq Q$ *then also* $P =_{\mathrm{L}} Q$.

PROOF. We prove that $P \models A$ iff $Q \models A$, by induction on the depth of A. For the formulas $B@n$ and $A_1 \triangleright A_2$ we need the congruence of $\simeq$. $\square$

The converse of Theorem 4.1 is the hard part of the characterisation. The crux of the proof is to find characteristic formulas for the capability and I/O primitives. Precisely, we look for formulas $\langle M \rangle . A$ (the *capability formulas*), where M is a capability, and $\langle !n \rangle$, $\langle ?n \rangle . A$ (the *I/O formulas*) with the following meaning:

- $P \models \langle \texttt{open}\, n \rangle . A$ iff there are P', P'' such that $P \equiv \texttt{open}\, n . P'$, $P' \Longrightarrow P''$, and $P'' \models A$
- $P \models \langle \texttt{in}\ n \rangle . A$ iff there are P', P'' such that $P \equiv \texttt{in}\, n . P'$, $P' \xRightarrow{(\text{out}\, n, \text{in}\, n)^\star} P''$, and $P'' \models A$
- $P \models \langle \texttt{out}\ n \rangle . A$ iff there are P', P'' such that $P \equiv \texttt{out}\, n . P'$, $P' \xRightarrow{(\text{in}\, n, \text{out}\, n)^\star} P''$, and $P'' \models A$
- $P \models \langle !n \rangle$ iff $P \equiv \{n\}$
- $P \models \langle ?n \rangle . A$ iff there are P', P'' such that $P \equiv (x)\ P'$, $\{n\} \mid (x)\ P' \Longrightarrow P''$, and $P'' \models A$

These formulas represent the temporal modalities that are observable in the intensional bisimilarity but are missing from the logic. With these formulas at hand, the converse of Theorem 4.1 will follow using standard arguments for modal logics.

A process P is *non-trivial* if not $P \equiv \mathbf{0}$. Thus P is non-trivial iff not $P \models \mathbf{0}$. A process P is *single* if it does not have two non-trivial parallel components at the outermost level; i.e., $P \equiv P'$ for some P' whose outermost operator is not a parallel composition. We will often use the characteristic formula of single processes:

$$\texttt{1comp} \stackrel{\text{def}}{=} \neg (\neg \mathbf{0} \mid \neg \mathbf{0})$$

To help understanding the definitions of the capability formulas, we first discuss some simpler formulas, which do not talk about the process underneath the prefix. We define, for names $n \neq h$:

$$\begin{array}{rcl}
\langle \texttt{open}\, n \rangle & \stackrel{\text{def}}{=} & n[\, h[\, \mathbf{0}\,]\,] \rhd \diamond\, (h[\, \mathbf{0}\,] \mid \mathbf{T}) \\
& & \wedge\ \texttt{1comp} \\
& & \wedge\ \neg \exists\, x\, .\ x[\, \mathbf{T}\,] \\
\langle \texttt{out}\ n \rangle & \stackrel{\text{def}}{=} & (\diamond\, (h[\, \mathbf{T}\,] \mid n[\, \mathbf{0}\,]))@n@h \\
& & \wedge\ \texttt{1comp} \\
& & \wedge\ \neg \exists\, x\, .\ x[\, \mathbf{T}\,]
\end{array}$$

It holds that $P \models \langle \texttt{open}\, n \rangle$ iff $P \equiv \texttt{open}\, n . P'$, for some P'. We sketch the proof. The subformula $\texttt{1comp} \wedge \neg \exists\, x\, .\ x[\, \mathbf{T}\,]$ says that P is single and is not an ambient. Thus, modulo $\equiv$, process P can only be $\mathbf{0}$, $\texttt{open}\, m . P'$ $\texttt{in}\, m . P'$, $\texttt{out}\, m . P'$, $(x)\ P'$, or $\{m\}$. The subformula $n[\, h[\, \mathbf{0}\,]\,] \rhd \diamond\, (h[\, \mathbf{0}\,] \mid \mathbf{T})$ says that $P \mid n[\, h[\, \mathbf{0}\,]\,]$ can reduce to a process with an empty ambient h at the outermost level. From these requirements, we conclude that $P \equiv \texttt{open}\, n . P'$, for some P'.

Similarly we prove that $P \models \langle \texttt{out}\ n \rangle$ iff $P \equiv \texttt{out}\, n . P'$, for some P'. By the subformula $\texttt{1comp} \wedge \neg \exists\, x\, .\ x[\, \mathbf{T}\,]$, process P is single and is not an ambient. By the subformula $(\diamond\, (h[\, \mathbf{T}\,] \mid n[\, \mathbf{0}\,]))@n@h$,

$$n[\, h[\, P\,]\,] \models \diamond\, (h[\, \mathbf{T}\,] \mid n[\, \mathbf{0}\,])$$

hence $P \equiv \texttt{out}\, n . P'$, for some P', otherwise $h[\, P\,]$ could not exit n.

To obtain the full capability formulas, namely $\langle \texttt{open}\, n \rangle . A$ and $\langle \texttt{out}\ n \rangle . A$, we add some quantification on names. Similarly we define $\langle \texttt{in}\ n \rangle . A$.

$$\begin{array}{rcl}
\langle \texttt{open}\, n \rangle . A & \stackrel{\text{def}}{=} & \forall\, x\, .\ \big(n[\, x[\, \mathbf{0}\,]\,] \rhd \diamond\, (x[\, \mathbf{0}\,] \mid A)\big) \\
& & \wedge\ \texttt{1comp} \\
& & \wedge\ \neg \exists\, x\, .\ x[\, \mathbf{T}\,] \\
\langle \texttt{out}\ n \rangle . A & \stackrel{\text{def}}{=} & \forall\, x\, .\ \big((\diamond\, (x[\, A\,] \mid n[\, \mathbf{0}\,]))@n@x\big) \\
& & \wedge\ \texttt{1comp} \\
& & \wedge\ \neg \exists\, x\, .\ x[\, \mathbf{T}\,] \\
\langle \texttt{in}\ n \rangle . A & \stackrel{\text{def}}{=} & \forall\, x\, .\ \big((n[\, \mathbf{0}\,] \rhd \diamond\, n[\, x[\, A\,]\,])@x\big) \\
& & \wedge\ \texttt{1comp} \\
& & \wedge\ \neg \exists\, x\, .\ x[\, \mathbf{T}\,]
\end{array}$$

The following formula is true of the *I/O processes*, that is, the messages and the abstractions.

$$\begin{array}{rcl}
\texttt{1comm} & \stackrel{\text{def}}{=} & \texttt{1comp} \wedge \neg\, \mathbf{0} \wedge \big(\neg \exists\, x\, .\ x[\, \mathbf{T}\,]\big) \\
& & \wedge\ \neg \big(\exists\, x\, .\ (\ \langle \texttt{open}\, x \rangle . \mathbf{T} \\
& & \qquad \vee\ \langle \texttt{out}\ x \rangle . \mathbf{T} \\
& & \qquad \vee\ \langle \texttt{in}\ x \rangle . \mathbf{T})\ \big)
\end{array}$$

The difficult part, however, is the definition of the I/O formulas, for separating messages from abstractions, but also, within the messages and the abstractions, messages with different contents and abstractions with different behaviours. The capability formulas are easier to define than the I/O formulas because capabilities act on ambients, and the logic has a connective, $n[\, A\,]$, for talking about ambients. By contrast, the I/O primitives act on themselves.

We define the I/O formulas proceeding as follows. First, we define a formula, `TestComm`, that characterises the special abstraction $(x)\ x[\, \mathbf{0}\,]$. Then we use `TestComm` to define the formulas for messages:

$$\langle !n \rangle \stackrel{\text{def}}{=} \texttt{1comm} \wedge (\texttt{TestComm} \rhd \diamond\, n[\, \mathbf{0}\,])$$

Finally, we then use $\langle !n \rangle$ to define the formulas for abstractions:

$$\langle ?n \rangle . A \stackrel{\text{def}}{=} \texttt{1comm} \wedge (\neg \exists x . \langle !x \rangle) \wedge (\langle !n \rangle \rhd \Diamond A)$$

We are therefore left with the definition of TestComm. This formula will actually be satisfied not only by $(x)\, x[\,\mathbf{0}\,]$, but by the set $\mathcal{S}$ of all abstractions that are beta*-convertible to $(x)\, x[\,\mathbf{0}\,]$, where beta*-conversion is the smallest congruence given by the axiom

$$\{M\} \mid (x)\, \eta[\,P\,] = \eta[\,P\,]\{M/x\} \qquad (\text{beta}^*)$$

Let:

$$\begin{array}{ll} \texttt{TestComm} & \stackrel{\text{def}}{=} \\ & \texttt{1comm} \\ & \wedge\ \texttt{1comm} \rhd \Box((\texttt{1comm} \mid \texttt{1comm}) \vee \exists x . x[\,\mathbf{0}\,]) \\ & \wedge\ \texttt{1comm} \blacktriangleright \Diamond\, n[\,\mathbf{0}\,] \\ & \wedge\ \texttt{1comm} \blacktriangleright \Diamond\, m[\,\mathbf{0}\,] \\ \text{where } n \text{ and } m \text{ are different names} & \end{array}$$

LEMMA 4.2. *It holds that $P \models \texttt{TestComm}$ iff $P \in \mathcal{S}$ and is closed.*

PROOF. (Sketch) We consider the implication from left to right (the opposite is easier). Since $P \models \texttt{1comm}$, process P must be an I/O. Also, P cannot be a message, otherwise it would not satisfy the formula

$$\texttt{1comm} \rhd \Box((\texttt{1comm} \mid \texttt{1comm}) \vee \exists x . x[\,\mathbf{0}\,])$$

This because a message in parallel with $(x)\,\mathbf{0}$ (that satisfies $\texttt{1comm}$) can reduce to $\mathbf{0}$, which does not satisfy $(\texttt{1comm} \mid \texttt{1comm}) \vee \exists x . x[\,\mathbf{0}\,]$.

We conclude that P should be an abstraction, say $(x)\, Q$. Now, from the formula $\texttt{1comm} \blacktriangleright \Diamond\, n[\,\mathbf{0}\,]$ and the formula $\texttt{1comm} \blacktriangleright \Diamond\, m[\,\mathbf{0}\,]$, we get that there are n_1 and n_2 such that

$$\begin{array}{l} \{n_1\} \mid (x)\, Q \models \Diamond\, n[\,\mathbf{0}\,] \\ \{n_2\} \mid (x)\, Q \models \Diamond\, m[\,\mathbf{0}\,] \end{array}$$

Finally we prove that these properties plus the formula

$$\texttt{1comm} \rhd \Box((\texttt{1comm} \mid \texttt{1comm}) \vee \exists x . x[\,\mathbf{0}\,])$$

only hold if $(x)\, Q \in \mathcal{S}$. □

There is an obvious correspondence between the clauses of $\simeq$ and the derived logical connectives for prefixes and I/O obtained above plus the primitive connectives $A \mid B$, $\mathbf{0}$, $n[\,A\,]$. This correspondence allows to prove the completeness of the logic. The proof also uses the n-th approximants $\simeq'_i$ of the bisimilarity $\simeq'$, (defined in the standard way from the definition of $\simeq'$) and the fact that $\bigcap_i \simeq'_i$ is the same as $\simeq'$.

THEOREM 4.3. *If $P =_{\mathrm{L}} Q$ then $P \simeq Q$.*

5. BARBED CONGRUENCE AND STRUCTURAL CONGRUENCE

Exploiting the co-inductive characterisation, we can now carry out the comparisons of $=_{\mathrm{L}}$ with $\approx$ and $\equiv$.

THEOREM 5.1. *Relation $=_{\mathrm{L}}$ is strictly included in $\approx$.*

The inclusion is proved from $=_{\mathrm{L}} \subseteq \simeq$ (Theorem 4.3) and $\simeq \subseteq \approx$ (essentially a consequence of the congruence of $\simeq$). The strictness of the inclusion is proved by these laws, that are valid for $\approx$ but not for $=_{\mathrm{L}}$:

1. $M.\, P = M \mid P$, where M is a capability, if P is of the form $M. \cdots . M. \mathbf{0}$
2. $(x)\, P = (x)\, \mathbf{0} \mid P$, if P is of the form $(x) \cdots (x)\, \mathbf{0}$
3. $(x)\, \{x\} = \mathbf{0}$

The third axiom is typical of behavioural equivalences of calculi where communication is asynchronous. The first and second axiom schemata are not valid for $=_{\mathrm{L}}$ because the logic allows us to intensionally observe parallel compositions. Here are two concrete examples of equalities derived from these axiom schemata:

$$\begin{array}{rcl} \texttt{in}\, n. \texttt{in}\, n & = & \texttt{in}\, n \mid \texttt{in}\, n \\ (x)\, (y)\, \mathbf{0} & = & (x)\, \mathbf{0} \mid (y)\, \mathbf{0} \end{array}$$

We do not know whether the laws (1-3) are all the laws that make $=_{\mathrm{L}}$ finer than $\approx$.

It turns out that $=_{\mathrm{L}}$ is almost the same as $\equiv$. The only difference is captured by the following axiom, the MA version of the familiar eta law of the λ-calculus:

$$(x)\, ((x)\, P \mid \{x\}) = (x)\, P \qquad (\text{eta})$$

We write $\equiv_{\mathrm{E}}$, and call it *normalised structural congruence*, for the relation defined with the rules of $\equiv$ plus eta. A process P is *eta-normalised* if no process P' with $P' \equiv P$ contains an eta redex $(x)((x)Q \mid \{x\})$. We write $\texttt{messages}(R)$ for the number messages in R, and $\texttt{pref}(R)$ for the number of prefixes in R.

LEMMA 5.2. *Suppose that $P \simeq Q$, and that both P and Q are eta-normalised. Then*

1. $\texttt{messages}(P) = \texttt{messages}(Q)$,
2. $\texttt{pref}(P) = \texttt{pref}(Q)$.

PROOF. The proofs of the two assertions are similar. We use the characterisation of $\simeq$ as $\simeq'$. For instance, for (1) we show that $\texttt{messages}(P) > \texttt{messages}(Q)$ would lead to contradiction, proceeding by induction on the shape of P. □

COROLLARY 5.3. *Suppose that $P \simeq Q$, and that both P and Q are eta-normalised. If $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q' \simeq P'$. Similarly, if $P \longrightarrow P'$, then there is Q' such that $Q \longrightarrow Q' \simeq P'$.*

PROOF. If Q performed more than one actions then Q would consume one more prefix or message than P, which, by Lemma 5.2, would lead to a contradiction. $\square$

LEMMA 5.4. *Suppose h does not appear in P. Then we have $P\{h/x\} \equiv Q\{h/x\}$ iff $P \equiv Q$.*

LEMMA 5.5. *Suppose that $P \simeq Q$, and that both P and Q are eta-normalised. Then $P \equiv Q$.*

PROOF. By induction on the shape of P, and using the characterisation of $\simeq$ as $\simeq'$. We show two example cases.

- $P = \mathtt{open}\, n.\, P'$. By definition of $\simeq'$, $Q \equiv \mathtt{open}\, n.\, Q'$ and there is Q'' such that $Q' \Longrightarrow Q'' \simeq' P'$. By Corollary 5.3, it must be $Q' = Q''$, therefore by induction $Q' \equiv P'$. We conclude that $P \equiv Q$.
- $P = (x)\, P'$. Then $Q \equiv (x)\, Q'$, for some Q'. Since $\simeq'$ is a congruence, $\{h\} \mid P \simeq' \{h\} \mid (x)\, Q'$. We have $\{h\} \mid P \longrightarrow P'\{h/x\}$, hence by Corollary 5.3, $\{h\} \mid (x)\, Q' \longrightarrow Q'\{h/x\} \simeq' P'\{h/x\}$. By induction, $P'\{h/x\} \equiv Q'\{h/x\}$, hence by Lemma 5.4, also $P' \equiv Q'$, therefore also $P \equiv Q$.

$\square$

THEOREM 5.6. *On closed MA processes, relations $=_{\mathrm{L}}$ and $\equiv_{\mathrm{E}}$ coincide.*

PROOF. Proving that $\equiv_{\mathrm{E}}$ coincides with $\simeq$. One inclusion is given by Theorem 4.3, the previous lemma and the validity of the eta law for $\simeq$. The converse holds too because all axioms defining $\equiv_{\mathrm{E}}$ are valid for $\simeq$. $\square$

6. EXTENSIONS OF THE CALCULUS

We consider some important extensions of MA. We do not change, however, the logic and the definition of satisfaction. We consider each feature separately. We have not studied combinations of the features; we believe, however, that this does not introduce new problems.

6.1 Capabilities in messages

In the original MA calculus [5], messages can also carry paths of capabilities. To accommodate this in the grammar of Table 1, all occurrences of η are replaced by M, and the path productions

$$M ::= M_1.\, M_2 \mid \epsilon\,,$$

are added to those for expressions, where ϵ stands for the empty path. Thus a capability can be a path, such as $\mathtt{open}\, n.\, \mathtt{in}\, m.\, \mathtt{open}\, h$. Also, the rules

$$\begin{aligned} (M_1.\, M_2).\, P &\equiv M_1.\, M_2.\, P \\ \epsilon.\, P &\equiv P \end{aligned}$$

are added to those of $\equiv$. Since messages can now carry names or capabilities, a type system [7] avoids run-time errors.

The results we have presented in the paper continue to hold under the extension. The most important modification is the definition of the characteristic formulas for messages carrying capabilities, in the proof of Theorem 4.3. For this, we need a formula `TestCap`, the analogous of the formula `TestComm` of Section 4, satisfied by all abstractions that are beta*-convertible to $(x)m[x.\,\mathbf{0}]$ where m is some fixed name.

We also need a formula $(\!|M|\!)$, for any closed capability M, that identifies those processes that are structurally congruent to $M.\,\mathbf{0}$. We first discuss an example, namely the formula $(\!|\mathtt{in}\, n.\, \mathtt{open}\, m|\!)$. For this, $\langle \mathtt{in}\ n\rangle.\, \langle \mathtt{open}\, m\rangle.\, \mathbf{0}$ is not enough: this formula is satisfied by $\mathtt{in}\, n.\, \mathtt{open}\, m.\, \mathbf{0}$ but also, for instance, by processes such as $\mathtt{in}\, n.\, (\{M\} \mid (x)\, \mathtt{open}\, m)$, which has some additional I/O, or $\mathtt{in}\, n.\, \mathtt{out}\, n.\, \mathtt{in}\, n.\, \mathtt{open}\, m.\, \mathbf{0}$, which stutters. We therefore set $(\!|\mathtt{in}\, n.\, \mathtt{open}\, m|\!)$ to be (the actual definition of $(\!|\mathtt{in}\, n.\, \mathtt{open}\, m|\!)$ will be syntactically different; the formula below is easier to read and semantically equivalent)

$$\begin{aligned} &\langle \mathtt{in}\ n\rangle.\, \langle \mathtt{open}\, m\rangle.\, \mathbf{0} \\ &\wedge \neg\, \langle \mathtt{in}\ n\rangle.\, \neg\, \mathtt{1comp} \\ &\wedge \neg\, \mathtt{in}\, n.\, \mathtt{out}\, n.\, \mathbf{T} \\ &\wedge \neg\, \langle \mathtt{in}\ n\rangle.\, \langle \mathtt{open}\, m\rangle.\, \neg\, \mathbf{0} \end{aligned}$$

The second, third and fourth $\wedge$-components take care of the problems with I/O and stuttering mentioned above. Here is the complete definition of the formulas $(\!|M|\!)$:

$$\begin{array}{rcl} (\!|\mathtt{open}\, n.\, M|\!) & \stackrel{\text{def}}{=} & \langle \mathtt{open}\, n\rangle.\, (\!|M|\!) \\ & & \wedge\, \neg\langle \mathtt{open}\, n\rangle.\, (\neg \mathtt{1comp} \vee \exists\, x\, .\, x[\,\mathbf{T}\,]) \\ (\!|\mathtt{out}\, n.\, M|\!) & \stackrel{\text{def}}{=} & \langle \mathtt{out}\ n\rangle.\, (\!|M|\!) \\ & & \wedge\, \neg\langle \mathtt{out}\ n\rangle.\, (\neg \mathtt{1comp} \vee \exists\, x\, .\, x[\,\mathbf{T}\,]) \\ & & \wedge\, \neg\langle \mathtt{out}\ n\rangle.\, \langle \mathtt{in}\ n\rangle.\, \langle \mathtt{out}\ n\rangle.\, (\!|M|\!) \\ (\!|\mathtt{in}\, n.\, M|\!) & \stackrel{\text{def}}{=} & \langle \mathtt{in}\ n\rangle.\, (\!|M|\!) \\ & & \wedge\, \neg\langle \mathtt{in}\ n\rangle.\, (\neg \mathtt{1comp} \vee \exists\, x\, .\, x[\,\mathbf{T}\,]) \\ & & \wedge;\neg\langle \mathtt{in}\ n\rangle.\, \langle \mathtt{out}\ n\rangle.\, \langle \mathtt{in}\ n\rangle.\, (\!|M|\!) \\ (\!|\epsilon.\, M|\!) & \stackrel{\text{def}}{=} & (\!|M|\!) \\ (\!|\mathbf{0}|\!) & \stackrel{\text{def}}{=} & \mathbf{0} \end{array}$$

In the definition of $(\!|M|\!)$, the formula $\neg\mathtt{1comp} \vee \exists\, x\, .\, x[\,\mathbf{T}\,]$ is used to control process reductions, see Lemma 6.1.

LEMMA 6.1. *Suppose $P \longrightarrow P'$. Then $P \models \neg\mathtt{1comp} \vee \exists\, x\, .\, x[\,\mathbf{T}\,]$.*

LEMMA 6.2. *Suppose M and P are closed. Then $P \models (\!|M|\!)$ iff $P \equiv M.\,\mathbf{0}$.*

PROOF. Induction on the depth of M. $\square$

We now define `TestCap`:

$$\begin{array}{rl}
\texttt{TestCap} \stackrel{\text{def}}{=} & \texttt{1comm} \\
& \wedge\ \texttt{1comm} \triangleright \Box(\ (\texttt{1comm} \mid \texttt{1comm}) \\
& \qquad\qquad \vee\ m[\,\texttt{1comp}\,]\ ) \\
& \wedge\ \texttt{1comm} \blacktriangleright \Diamond\ m[\,(\!|\texttt{in}\, n|\!)\,] \\
& \wedge\ \texttt{1comm} \blacktriangleright \Diamond\ m[\,\mathbf{0}\,]
\end{array}$$

where n and m are different names.

The correctness of this definition is proved along the lines of that of `TestComm`. The formula $\langle !M \rangle$, where M is any closed capability, is then

$$\langle !M \rangle \stackrel{\text{def}}{=} \texttt{1comm} \wedge \big(\texttt{TestCap} \triangleright \Diamond\ m[\,(\!|M|\!)\,]\big)$$

6.2 Synchronous Ambients

Since the modal logic does not talk about the I/O primitives, it is interesting to examine variations of these primitives, to see the effect on the equality induced by the logic. In MA communication is asynchronous: since a message has no continuations, no process is blocked until the message is consumed. The most natural variation consists in making communication *synchronous*. For this the production $\{\eta\}$ for messages in the grammar of MA in Table 1 is replaced by the production $\{\eta\}.\,P$. Reduction rule `Red-Com` becomes:

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

The communication act liberates, at the same time, both the continuation P of the abstraction *and* the continuation Q of the message. We write $\mathrm{MA}^{\mathrm{sync}}$ for the resulting synchronous calculus.

Synchrony leads to some important modifications in the assertions and in the proofs of the results in the paper. In $\mathrm{MA}^{\mathrm{sync}}$, the eta law fails and, surprisingly, the equality induced by the logic is precisely structural congruence:

THEOREM 6.3. *On closed $\mathrm{MA}^{\mathrm{sync}}$ processes, relations $=_{\mathrm{L}}$ and $\equiv$ coincide.*

The co-inductive characterisation in terms of $\simeq$ (that is, Theorems 4.1 and 4.3) remains true, provided that in the definition of the bisimilarity the 'asynchronous' clause (8) for abstractions is replaced by the clause:

- If $P \xrightarrow{?M} P'$ then there is Q' such that $Q \stackrel{?M}{\Longrightarrow} Q'$ and $P' \simeq Q'$

The proof of the characterisation, however, is harder. In asynchronous MA, our separation of messages from abstractions exploited their asymmetry: abstractions, but not messages, have a continuation. In the synchronous case the asymmetry disappears, therefore we have to use a different route for the proof. Again, the most delicate point is to find a replacement for the formula `TestComm`. We only quickly sketch how the new definition is obtained.

We first define a formula, `OnlyCom`, that is satisfied only by abstractions $(x)\,P$ and messages $\{M\}.\,P$ in which capability prefixes and ambients do not appear in the continuation P and, moreover, no subterm of P contains more than two non-trivial parallel components:

$$\begin{array}{rl}
\texttt{OnlyCom} \stackrel{\text{def}}{=} & \texttt{1comm} \blacktriangleright (\ \Box((\texttt{1comm} \mid \texttt{1comm}) \vee \mathbf{0}) \\
& \qquad\qquad \wedge \Diamond\ \mathbf{0})
\end{array}$$

Using `OnlyCom` we define a formula, `ComAmb`, that is satisfied only by processes defined as those that satisfy `OnlyCom` except that the innermost operator is an ambient $\eta[\,y[\,\mathbf{0}\,]\,]$:

$$\begin{array}{l}
\texttt{ComAmb} \stackrel{\text{def}}{=} \\
\quad \exists\, x\,.\ \big(\ \texttt{OnlyCom} \blacktriangleright \big(\ \Box(\ (\texttt{1comm} \mid \texttt{1comm})\ \vee\ x[\,n[\,\mathbf{0}\,]\,]) \\
\qquad\qquad\qquad\qquad\qquad \wedge \Diamond\ x[\,n[\,\mathbf{0}\,]\,]\big) \\
\qquad\quad \wedge\ \texttt{OnlyCom} \blacktriangleright \Diamond\ x[\,m[\,\mathbf{0}\,]\,]\big)
\end{array}$$

where n and m are different names.

We then define a formula that characterises the abstraction $(x)\ h[\,x[\,\mathbf{0}\,]\,]$; we write `3comm` for `1comm` | `1comm` | `1comm`:

$$\begin{array}{rl}
\texttt{Imm}_h \stackrel{\text{def}}{=} & \texttt{ComAmb} \\
& \wedge\ \texttt{OnlyCom} \triangleright (\Box\neg\,\texttt{1comm} \wedge \Box\neg\,\texttt{3comm}) \\
& \wedge\ \texttt{OnlyCom} \blacktriangleright \Diamond\ h[\,n[\,\mathbf{T}\,]\,] \\
& \wedge\ \texttt{OnlyCom} \blacktriangleright \Diamond\ h[\,m[\,\mathbf{T}\,]\,]
\end{array}$$

where n and m are different names.

Roughly, the first $\wedge$-component implies that a process that satisfies $\texttt{Imm}_h$ has an abstraction or a message as the outermost operator, and an ambient $\eta[\,x[\,\mathbf{0}\,]\,]$ as the innermost. The second $\wedge$-component, say A, ensures that the process does not have any other operators; that is, the ambient $\eta[\,x[\,\mathbf{0}\,]\,]$ is reached immediately after the initial communication. For instance, the process $R \stackrel{\text{def}}{=} \{M\}.\,(x)\ h[\,x[\,\mathbf{0}\,]\,]$ does not satisfy A because $R \mid (x)\,\mathbf{0} \Longrightarrow (x)\ h[\,x[\,\mathbf{0}\,]\,]$ and $(x)\ h[\,x[\,\mathbf{0}\,]\,]$ satisfies `1comm`. Finally, the third and fourth $\wedge$-components rule out the messages and the abstraction $(x)\ x[\,x[\,\mathbf{0}\,]\,]$.

Now we can define the formula:

$$\begin{array}{rl}
\langle !n \rangle.\,A \stackrel{\text{def}}{=} & \texttt{1comm} \\
& \wedge\ \forall\, x\,.\ (\texttt{Imm}_x \triangleright \Diamond\ (x[\,n[\,\mathbf{0}\,]\,] \mid A))
\end{array}$$

LEMMA 6.4. *Suppose P is a closed $\mathrm{MA}^{\mathrm{sync}}$ process. It holds that $P \models \langle !n \rangle.\,A$ iff there are Q and Q' such that $P \equiv \{n\}.\,Q$, and $Q \Longrightarrow Q'$, and $Q' \models A$.*

And finally we can define the formulas for abstractions:

$$\begin{aligned}\langle ?n\rangle . A \quad &\stackrel{\text{def}}{=} \quad \mathtt{1comm} \wedge \neg\exists\, x\,.\, \langle !x\rangle . \mathbf{T} \\ & \qquad \wedge \forall\, x\,.\, \big(\langle !n\rangle . x[\,\mathbf{0}\,] \rhd \Diamond\,(x[\,\mathbf{0}\,] \mid A)\big)\end{aligned}$$

LEMMA 6.5. *Suppose P is a MA^{sync} process. It holds that $P \models \langle ?n\rangle . A$ iff there are P' and P'' such that $P \equiv (x)\, P'$, and $P'\{n/x\} \Longrightarrow P''$, and $P'' \models A$.*

6.3 Recursion

The final extension we consider is the addition of recursive definitions $\mathtt{rec}\ X.\, P$ to the process grammar. To avoid infinite nesting of ambients, in $\mathtt{rec}\ X.\, P$ the recursion variable X is unboxed, i.e., it may not occur free inside an ambient. To facilitate the analysis, we also require X to be guarded, i.e. X may occur free only underneath a prefix or an abstraction; we leave for future work whether this restriction can be lifted. (Using unguarded recursion, the full replication operator, $\mathtt{bang}(P)$, is definable thus: $\mathtt{bang}(P) \stackrel{\text{def}}{=} \mathtt{rec}\ X.\,(X \mid P)$.) In the operational semantics, the fold/unfold rule

$$\mathtt{rec}\ X.\, P \equiv P\{\mathtt{rec}\ X.\, P/X\}$$

is added to those of structural congruence. We write MA^{rec} for this extension of MA.

The co-inductive characterisation, Theorems 4.1 and 4.3, remains true, under the condition that processes are image-finite. (An MA process is image-finite if it has only a finite number of different derivatives under each transition relation $\stackrel{\mu}{\Longrightarrow}$ and under the stuttering relations $P \xRightarrow{(\mathtt{in}\, n, \mathtt{out}\, n)^\star} P'$ and $P \xRightarrow{(\mathtt{out}\, n, \mathtt{in}\, n)^\star} P'$. Image-finiteness is a standard condition for characterisations of operational equivalences on infinite processes using finitary logics.)

THEOREM 6.6. *On image-finite MA^{rec} processes, relations $\simeq$ and $=_{\text{L}}$ coincide.*

We have pointed out that the stuttering relations can be omitted from the definition of the bisimilarity $\simeq$ in the finite calculus of Table 1. We show why stuttering is needed in the calculus with recursion. These processes are behaviourally equivalent, and are also equal in the logic, but are distinguished by $\simeq$ without stuttering:

$$\begin{aligned} P &\stackrel{\text{def}}{=} \mathtt{in}\, n.\, \mathtt{rec}\ X.\,(\{M\} \mid (x)\,(\mathtt{out}\, n.\, \mathtt{in}\, n.\, X)) \\ Q &\stackrel{\text{def}}{=} \mathtt{in}\, n.\, \mathtt{rec}\ X.\,(\mathtt{out}\, n.\, \mathtt{in}\, n.\,(\{M\} \mid (x)\, X)) \end{aligned}$$

The recursions in the processes implement a loop with an alternation between two I/O and the path $\mathtt{out}\, n.\, \mathtt{in}\, n$. There is a 1-cycle shift, however, between the two loops. Stuttering makes the shift irrelevant.

As a consequence of Theorem 6.6, we have:

THEOREM 6.7. *On image-finite MA^{rec} processes, $=_{\text{L}}$ is strictly included in $\approx$.*

Having recursion, however, the intensionality of logic is stronger. Here are two examples of behavioural equalities that are not valid in the logic. We write $\mathtt{bang}(M)$ for the process $\mathtt{rec}\, X.\, M.\, X$, and $\mathtt{bang}((x)\, P)$ for $\mathtt{rec}\, X.\,(x)\,(X \mid P)$.

$$\begin{aligned} \mathtt{bang}(M_1) \mid M_2.\, M_1 \quad &= \quad \mathtt{bang}(M_1) \mid M_2 \\ n[\,(x)\, P\,] \mid \mathtt{bang}((x)\, P) \quad &= \quad n[\,\mathbf{0}\,] \mid \mathtt{bang}((x)\, P) \end{aligned}$$

A challenging problem for future work is to find a proof system for $=_{\text{L}}$ in the calculus with recursion, extending our axiomatisation of the finite calculus.

7. REFERENCES

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

[2] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[3] 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.

[4] L. Cardelli and G. Ghelli. A query language for semistructured data based on the ambient logic. submitted, April 2000.

[5] 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.

[6] L. Cardelli and A.D. Gordon. Technical annex to [5]. Unpublished notes, 1998.

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

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

[9] R. De Nicola and R. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

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

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

[12] F. Levi and D. Sangiorgi. Controlling interference in ambients. To appear as a Technical Report. Short version appeared in *Proc. 27th POPL*, ACM Press, 2000.

[13] 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.

[14] 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.