Basic Observables for a Calculus for Global Computing

Rocco De Nicola Daniele Gorla Rosario Pugliese
Dipartimento di Sistemi e Informatica, Università di Firenze
email : {denicola, gorla, pugliese}@dsi.unifi.it

December 1, 2004

Abstract

We discuss a basic process calculus useful for modelling applications over global computing systems and present the associated semantic theories as determined by some basic notions of observation. The main features of the calculus are explicit distribution, remote operations, process mobility and asynchronous communication through distributed data spaces. We introduce some natural notions of extensional observations and study their closure under operational reductions and/or language contexts to obtain *barbed congruence* and *may testing*. For these equivalences, we provide alternative tractable characterizations as labelled bisimulation and trace equivalence. We discuss some of the induced equational laws and relate them to design choices of the calculus. In particular, we show that some of these laws do not hold any longer if the language is rendered less abstract by introducing (asynchronous and undetectable) failures or by implementing remote communications via process migrations and local exchanges. In both cases, we also investigate the adaptation of the tractable characterizations of barbed congruence and may testing to the lower-level scenarios.

Contents

1 Introduction

Programming computational infrastructures available globally for offering uniform services has become one of the main issues in Computer Science. The challenges come from the variable guarantees for communication, co-operation and mobility, resource usage, security policies and mechanisms, etc. that have to be taken into account. A key issue is the definition of innovative theories, computational paradigms, linguistic mechanisms and implementation techniques for the design, realisation, deployment and management of global computational environments and their application.

On the foundational side, the demand is on the development of tools and techniques to build safer and trustworthy global systems, to analyse their behaviour, and to demonstrate their conformance to given specifications. Indeed, theoretical models and calculi can provide a sound basis for building systems which are "sound by construction" and which behave in a predictable and analysable manner. The crux is to identify what abstractions are more appropriate for programming global computers and to supply effective tools to support development and certification of global computing applications. This paper should be considered as a contribution to this line of research.

A distinguishing feature of applications over so-called "global computers" is the necessity of dealing with dynamic and unpredictable changes of their network environment, due to unavailability of network connectivity, bandwidth fluctuations, lack of resources, failure of nodes, network reconfigurations, etc.. These issues have to be considered together with the more traditional ones of distributed applications, like heterogeneity (of operating systems and application software), scalability (huge number of users and nodes) and autonomy (resources managed by different administration domains). Indeed, global computers are fostering a new style of distributed programming whose key principle is *network awareness*, i.e. applications have information about the network (location, latency, congestion, etc.) and can adapt to its variations. Moreover, applications should also support *disconnected operations* [37], that permit software components to be remotely executed even if their owner is not connected.

In our view, a language for global computing should be equipped with primitives that support:

network awareness, i.e. locations can be explicitly referenced and operations can be remotely invoked;

disconnected operations, i.e. code can be moved from one location to the other and remotely executed;

flexible communication mechanisms, like distributed repositories [18, 17, 23] storing content addressable data;

remote operations, like asynchronous remote communications.

Semantic theories, needed for stating and proving observable properties, should reflect all the above listed distinctive features of global systems at user's application level, but they should ignore issues such as routing or network topology, because they are hardly observable at the user level.

Several foundational languages, presented as process calculi or strongly based on them, have been developed that have improved the formal understanding of the complex mechanisms underlying global computers. We want to mention the Ambient calculus [13], Dπ [28], DJoin [24] and Nomadic Pict [41]. They are equipped with primitives to represent at various abstraction levels the execution contexts of the net where applications roam and run, they provide mechanisms for coordinating and monitoring the use of resources, and they support the specification and the implementation of security policies. However, if one contrasts them with the above list of distinguishing features of languages for global computers, one realizes that all of them fall short for at least one of the targets.

Here we want to develop the semantic theory of an alternative model that takes its origin from two formalisms with opposite objectives, namely the programming language X-KLAIM [6], a full fledged programming language based on KLAIM [19], and the π-calculus [33], the generally recognized minimal common denominator of calculi for mobility. The resulting model has been called cKLAIM (*core* KLAIM)[1]. It can be thought of as a variant of the π-calculus with process distribution and mobility, remote operations and asynchronous communication through distributed repositories. Given the direct correspondence of cKLAIM with X-KLAIM, we believe that the tractable behavioural equivalences we develop in this paper provide powerful tools to write sound programs for global computers. First, programs written in X-KLAIM are mapped down to cKLAIM; here they are verified, by using behavioural equivalences to formalize and prove properties; finally, they can run on an actual global computer, like the Internet, by exploiting their Java-based translation [7].

We develop the semantic theory of the proposed language by defining behavioural equivalences over terms as the maximal congruences induced by some basic *observables* that are dictated by the relevant features of global computers. The approach can be summarized as follows:

1. Define a set of observables (values, normal forms, actual communications, ...) to which a term can evaluate by means of successive reductions.

2. Define a basic equivalence over terms by stating that two terms are equivalent if and only if they exhibit the same set of basic observables.

3. Consider the largest congruence over the language induced by the basic equivalence or by its co-inductive closure.

A similar approach has already been used to study models of concurrent systems (e.g., CCS [34, 10] and π-calculus [40, 4]). Obviously, the designation of basic observables is critical. Thus, we draw inspiration from everyday experience: a user can observe the behaviour of a global computer (at the application level) by testing

i. whether a specific site is up and running (i.e., it provides some data of any kind),

ii. whether a specific information is present in (at least) a site, or

iii. whether a specific information is present in a specific site.

Other calculi for global computers rely on (barbed) congruences induced by similar observables: for example, Ambient [13] uses a barb that is somehow related to *i*. above, while the barbs in Dπ-calculus [28] are strongly related to *iii*. .

A question that naturally arises is whether these observables yield ‘interesting’ congruences. The three basic observables, together with the discriminating power provided by cKLAIM contexts, all yield the same congruence, when used similarly. This is for us already an indication of the robustness of the resulting semantic theories. Moreover, as we will show, the observables are still sufficiently powerful to give rise to interesting semantic theories also when considering lower-level features like, e.g., failures. Due to its intuitive appeal, in the rest of this paper we shall use only the first kind of observable.

A major drawback of the approach relying on basic observables and context closures is that the resulting congruences are defined via universal quantification over all language contexts, and

[1]As a programming notation, cKLAIM was presented in [5]; here, we turn it into a calculus, by equipping it with an LTS-based operational semantics and a few behavioural equivalences.

Nets:	N	::=	$\mathbf{0}$ | $l :: C$ | $N_1 \parallel N_2$ | $(\nu l)N$
Components:	C	::=	$\langle l \rangle$ | P | $C_1 \mid C_2$
Processes:	P	::=	$\mathbf{nil}$ | $a.P$ | $P_1 \mid P_2$ | X | $\mathbf{rec}\, X.P$
Actions:	a	::=	$\mathbf{in}(p)@u$ | $\mathbf{out}(u_2)@u_1$ | $\mathbf{eval}(P)@u$ | $\mathbf{new}(l)$
Input Parameters:	p	::=	u | $!\, x$

Table 1: cKlaim Syntax

this makes their checking very hard. It is then important to devise proof techniques that avoid such quantification. We shall define a labelled transition system (with labels indicating the performed action) and exploit the labels to avoid quantification over contexts. We shall present tractable characterizations of two 'touchstone' congruences, namely *barbed congruence* and *may testing*, in terms of (non-standard) labelled *bisimilarity* and *trace* equivalence, respectively. In doing this, we have to face the problems raised by the presence of explicit localities and by the fact that cKlaim is asynchronous (both in the communication and in the mobility paradigm) and higher-order (because processes can migrate).

The rest of the paper is organized as follows. In Section 2 we present cKlaim's syntax and reduction-based semantics. In Section 3 we define barbed congruence and may testing, while in Sections 4 and 5 we present their alternative characterizations. In Section 7 we discuss some of the equations induced by the semantic theories and show that some of them break down when rendering the language less abstract by implementing remote communications via process migrations and local exchanges or by introducing (asynchronous and undetectable) failures. Finally, in Section 8, we discuss related work.

2 The Process Language cKlaim

In this section, we present the syntax of cKlaim and its operational semantics based on a structural congruence and a labelled transition system (LTS).

2.1 Syntax

The syntax of cKlaim is reported in Table 1. A countable set $\mathcal{N}$ of *names* $l, l', \ldots, u, \ldots, x, y, \ldots, X, Y, \ldots$ is assumed. Names provide the abstract counterpart of the set of *communicable* objects and can be used as localities, basic variables or process variables: we do not distinguish between these three kinds of objects. Notationally, we prefer letters $l, l', \ldots$ when we want to stress the use of a name as a locality, $x, y, \ldots$ when we want to stress the use of a name as a basic variable, and $X, Y, \ldots$ when we want to stress the use of a name as a process variable. We will use u for basic variables and localities.

Processes, ranged over by $P, Q, R, \ldots$, are the cKlaim active computational units and may be executed concurrently either at the same locality or at different localities. They are built up from the terminated process **nil** and from the basic actions by using prefixing, parallel composition and recursion. *Actions* permit removing/adding data from/to node repositories, activating new threads of execution and creating new nodes. Action **new** is not indexed with an address because it always acts locally; all the other actions explicitly indicate the (possibly remote) locality where they will take effect. Notice that $\mathbf{in}(l)@l'$ evolves only if datum $\langle l \rangle$ is present in l'; indeed, $\mathbf{in}(l)@l'$ is a form of *name matching operator* derived from the Linda [26] pattern-matching.

a	$fn()$	$bn()$
$\mathbf{in}(!\,x)@u$	$\{u\}$	$\{x\}$
$\mathbf{in}(u_2)@u_1$	$\{u_1, u_2\}$	$\emptyset$
$\mathbf{out}(u_2)@u_1$	$\{u_1, u_2\}$	$\emptyset$
$\mathbf{eval}(P)@u$	$fn(P) \cup \{u\}$	$bn(P)$
$\mathbf{new}(l)$	$\emptyset$	$\{l\}$

N	$fn()$	$bn()$
$\mathbf{nil}$	$\emptyset$	$\emptyset$
$l :: C$	$\{l\} \cup fn(C)$	$bn(C)$
$N_1 \parallel N_2$	$fn(N_1) \cup fn(N_2)$	$bn(N_1) \cup bn(N_2)$
$(\nu l)N$	$fn(N) - \{l\}$	$bn(N) \cup \{l\}$

C	$fn()$	$bn()$
$\mathbf{nil}$	$\emptyset$	$\emptyset$
$\langle l \rangle$	$\{l\}$	$\emptyset$
$a.P$	$(fn(P) - bn(a)) \cup fn(a)$	$bn(P) \cup bn(a)$
$P_1 \mid P_2$	$fn(P_1) \cup fn(P_2)$	$bn(P_1) \cup bn(P_2)$
X	$\{X\}$	$\emptyset$
$\mathbf{rec}\,X.P$	$fn(P) - \{X\}$	$bn(P) \cup \{X\}$

Table 2: Free and bound names

Nets, ranged over by $N, M, H, K, \ldots$, are finite collections of nodes. A *node* is a pair $l :: C$, where locality l is the address of the node and C is the (parallel) component located at l. *Components*, ranged over by $C, D, \ldots$, can be either processes or data, denoted by $\langle l \rangle$. In the net $(\nu l)N$, the scope of the name l is restricted to N; the intended effect is that if one considers the net $N_1 \parallel (\nu l)N_2$ then locality l of N_2 cannot be immediately referred to from within N_1 (this is the powerful π-calculus' mechanism for restricted names).

Names occurring in cKlaim processes and nets can be *bound*. More precisely, prefix $\mathbf{in}(!x)@u.P$ binds x in P; this prefix is similar to the λ-abstraction of the λ-calculus. Prefix $\mathbf{new}(l).P$ binds l in P, and, similarly, net restriction $(\nu l)N$ binds l in N. Finally, $\mathbf{rec}\,X.P$ binds X in P. A name that is not bound is called *free*. The sets $fn(\cdot)$ and $bn(\cdot)$ (respectively, of free and bound names of a term) are defined accordingly (their definitions are shown in Table 2). The set $n(\cdot)$ of names of a term is the union of its sets of free and bound names. As usual, we say that two terms are *alpha-equivalent*, written $\equiv_\alpha$, if one can be obtained from the other by renaming bound names. We shall say that a name u is fresh for $\_$ if $u \notin n(\_)$. In the sequel, we shall work with terms whose bound names are all distinct and different from the free ones.

Notation 2.1 We write $A \triangleq W$ to mean that A is of the form W; this notation is used to assign a symbolic name A to the term W. We shall use notation $\widetilde{\cdot}$ to denote tuples of objects (e.g. $\widetilde{l}$ is a tuple of names); this will be sometimes written as $\widetilde{x}_{i \in I}$, for an appropriate index-set I. Moreover, if $\widetilde{l} = (l_1, ..., l_n)$, we shall assume that $l_i \neq l_j$ for $i \neq j$. If $\widetilde{x} = (x_1, \ldots, x_n)$ and $\widetilde{y} = (y_1, \ldots, y_m)$ then $\widetilde{x}, \widetilde{y}$ will denote the tuple of pairwise distinct elements $(x_1, \ldots, x_n, y_1, \ldots, y_m)$. When convenient, we shall regard a tuple simply as a set (thus we can write e.g. $\widetilde{l} \subseteq L$ to mean that all components of $\widetilde{l}$ are in L). We shall sometimes write $\mathbf{in}()@l$, $\mathbf{out}()@l$ and $\langle\rangle$ to mean that the argument of the actions or the datum are irrelevant. Finally, we omit trailing occurrences of process $\mathbf{nil}$ and write $\prod_{j=1}^{n} W_j$ for the parallel composition (both '|' and '||') of terms (components or nets, resp.) W_j.

2.2 Operational Semantics

cKlaim operational semantics is given in terms of a structural congruence and a reduction relation. The *structural congruence*, $\equiv$, identifies nets which intuitively represent the same net. It is defined as the least congruence relation over nets that satisfies the laws in Table 3. Most of the laws are

(ALPHA)	N	$\equiv$	N'	if $N \equiv_\alpha N'$
(PZERO)	$N \parallel \mathbf{0}$	$\equiv$	N	
(PCOM)	$N_1 \parallel N_2$	$\equiv$	$N_2 \parallel N_1$	
(PASS)	$(N_1 \parallel N_2) \parallel N_3$	$\equiv$	$N_1 \parallel (N_2 \parallel N_3)$	
(RCOM)	$(\nu l_1)(\nu l_2)N$	$\equiv$	$(\nu l_2)(\nu l_1)N$	
(EXT)	$N_1 \parallel (\nu l)N_2$	$\equiv$	$(\nu l)(N_1 \parallel N_2)$	if $l \notin fn(N_1)$
(ABS)	$l :: C$	$\equiv$	$l :: (C\vert\mathbf{nil})$	
(REC)	$l :: \mathbf{rec}\, X.P$	$\equiv$	$l :: P[^{\mathbf{rec}\, X.P}/_X]$	
(RNODE)	$(\nu l)N$	$\equiv$	$(\nu l)(N \parallel l :: \mathbf{nil})$	
(CLONE)	$l :: C_1\vert C_2$	$\equiv$	$l :: C_1 \parallel l :: C_2$	

Table 3: Nets Structural Congruence

taken from the π-calculus (see, e.g., [38]) with law (ABS), that is the equivalent of law (PZERO) for '|', and law (REC), that freely folds/unfolds recursive definitions. Additionally, we have law (RNODE), saying that any restricted name can be used as the address of a node[2], and law (CLONE), that transforms a parallel between co-located components into a parallel between nodes. Notice that commutativity and associativity of '|' can be obtained by (PCOM), (PASS) and (CLONE). In the sequel, by exploiting Notation 2.1 and law (RCOM), we shall write $(\nu\widetilde{l})N$ to denote a net with a (possible empty) set $\widetilde{l}$ of restricted localities.

In what follows, we shall only consider nets where each bound name is associated to a node; by virtue of rule (RNODE) this is always possible.

The reduction relation is given in Table 4. In rules (R-OUT) and (R-EVAL), existence of the node target of the action is necessary to place the spawned component. Notice that existence of the target node can only be checked at run-time. Indeed, an approach like [28] does not fit well global computing setting because it relies on a typing mechanism that would require the knowledge of the whole net. Rules (R-IN) and (R-MATCH) require existence of a matching datum in the target node. Rule (R-MATCH) says that action $\mathbf{in}(l_2)@l_1$ consumes exactly the datum $\langle l_2 \rangle$ at l_1. Rule (R-IN) says that action $\mathbf{in}(!\,x)@l_1$ can consume any datum l_2 at l_1; l_2 will then replace the free occurrences of x in the continuation of the process performing the action. Rule (R-NEW) says that execution of action $\mathbf{new}(l')$ simply adds a restriction over l' to the net; from then on, a new node with locality l' can be allocated/deallocated by using law (RNODE).

cKLAIM adopts a LINDA-like [26] communication mechanism: communication is asynchronous and data are anonymous. Notice that, even if there exist prefixes for placing data to nodes, no synchronization takes place between (sending and receiving) processes. On the contrary, a sort of synchronization takes place between a sending process and its target node (see rules (R-OUT) and (R-EVAL)). A similar synchronization takes place between the node hosting a datum and the process looking for it (see rules (R-IN) and (R-MATCH)).

[2]Restricted names can be thought of as private network addresses, whose corresponding nodes can be activated when needed, and successively deactivated, by the owners of the resource (i.e. the nodes included in the scope of the restriction). If names would represent not only localities but also other communicable objects, the law should be slightly modified for it to deal only with bound locality names.

(R-Out)
$$l :: \mathbf{out}(l_2)@l_1.P \;\|\; l_1 :: \mathbf{nil} \;\longmapsto\; l :: P \,\|\, l_1 :: \langle l_2 \rangle$$

(R-Eval)
$$l :: \mathbf{eval}(P_2)@l_1.P_1 \;\|\; l_1 :: \mathbf{nil} \;\longmapsto\; l :: P_1 \,\|\, l_1 :: P_2$$

(R-In)
$$l :: \mathbf{in}(!x)@l_1.P \;\|\; l_1 :: \langle l_2 \rangle \;\longmapsto\; l :: P[^{l_2}/_x] \,\|\, l_1 :: \mathbf{nil}$$

(R-Match)
$$l :: \mathbf{in}(l_2)@l_1.P \;\|\; l_1 :: \langle l_2 \rangle \;\longmapsto\; l :: P \,\|\, l_1 :: \mathbf{nil}$$

(R-New)
$$l :: \mathbf{new}(l').P \;\longmapsto\; (\nu l')(l :: P \,\|\, l' :: \mathbf{nil})$$

(R-Par)
$$\frac{N_1 \;\longmapsto\; N_1'}{N_1 \,\|\, N_2 \;\longmapsto\; N_1' \,\|\, N_2}$$

(R-Res)
$$\frac{N \;\longmapsto\; N'}{(\nu l)N \;\longmapsto\; (\nu l)N'}$$

(R-Struct)
$$\frac{N \;\equiv\; M \;\longmapsto\; M' \;\equiv\; N'}{N \;\longmapsto\; N'}$$

Table 4: cKlaim Operational Semantics

3 Touchstone Equivalences

In this section we present (weak) equivalences yielding sensible semantic theories for cKlaim. The approach we follow relies on the definition of an *observation* (also called *barb*) that intuitively formalises the interactions a process can be engaged in. We use observables to define equivalences that equate those nets that cannot be taken apart by any basic observation (*i*) during their computations, or (*ii*) in any net context, or (*iii*) during their computations in any net context. Notationally, we shall use $\models\!\!\Rightarrow$ to denote the reflexive and transitive closure of $\longmapsto$.

Definition 3.1 (Barbs and Net Contexts)
Predicate $N \downarrow l$ holds true if and only if $N \equiv (\nu\widetilde{l})(N' \,\|\, l :: \langle l' \rangle)$ for some $\widetilde{l}$, N' and l' such that $l \notin \widetilde{l}$.

Predicate $N \Downarrow l$ holds true if and only if $N \models\!\!\Rightarrow N'$ for some N' such that $N' \downarrow l$.

A net context $C[\cdot]$ *is a* cKlaim *net with an occurrence of a hole* $[\cdot]$ *to be filled in with any net. Formally,*

$$C[\cdot] \quad ::= \quad [\cdot] \quad \Big| \quad N \,\|\, C[\cdot] \quad \Big| \quad (\nu l)C[\cdot]$$

We have chosen the basic observables by taking inspiration from the corresponding ones of the asynchronous π-calculus [4]. One may wonder if our choice is “correct” and argue that there are other alternative notions of basic observables that seem quite natural, as we have discussed in the Introduction. A first alternative could be to consider as equivalent two nets if they make available the same set of data, possibly in different nodes. A second alternative could be to consider as equivalent two nets if they have exactly the same data at the same localities. Later on, we shall prove that the congruences induced by these alternative observables do coincide. This means that our results are quite independent from the observable chosen and vindicates our choice. Moreover, notice that, by using other kinds of observation predicates, more sophisticated equivalences should come into the picture. For example, in [10] it is shown how *must testing* and *fair testing* can be obtained in CCS by only changing the basic observable.

Now, we say that a binary relation $\mathfrak{R}$ between nets is

- *barb preserving*, if $N \,\mathfrak{R}\, M$ and $N \Downarrow l$ imply $M \Downarrow l$;
- *reduction closed*, if $N \,\mathfrak{R}\, M$ and $N \longmapsto N'$ imply $M \models\!\!\Rightarrow M'$ and $N' \,\mathfrak{R}\, M'$;
- *context closed*, if $N \,\mathfrak{R}\, M$ implies $C[N] \,\mathfrak{R}\, C[M]$ for every context $C[\cdot]$.

Our touchstone equivalences should at the very least relate nets with the same observable behaviour; thus, they must be barb preserving. However, an equivalence defined only in terms of this property can be hardly considered a 'touchstone': indeed, the set of barbs changes during computations (usually, it shrinks) or when interacting with an external environment (usually, it widens). Moreover, for the sake of compositionality, our touchstone equivalences should also be congruences. These requirements lead us to the following definitions.

Definition 3.2 (May testing) $\simeq$ *is the largest symmetric, barb preserving and context closed relation between nets.*

Definition 3.3 (Barbed congruence) $\cong$ *is the largest symmetric, barb preserving, reduction and context closed relation between nets.*

We want to remark that the above definition of barbed congruence is the standard one, see [29, 38]. May testing is, instead, usually defined in terms of *observers*, *experiments* and possible *successes of experiment* [22]. However, if we let $\simeq'$ denote the equivalence on cKlaim nets defined a là [22], we can prove that the two definitions do coincide. Moreover, the inclusions between our touchstone equivalences reflect the inclusions that hold in the π-calculus. To define may testing like in [22], we let `test` be a fresh and reserved name used to report *success* of an *experiment* (i.e. a computation) of a net and an *observer*. The latter is a net containing (*i*) a node whose address is `test` that can only host the datum $\langle$`test`$\rangle$, and (*ii*) processes that may emit the datum $\langle$`test`$\rangle$ at `test`. A computation reports success if, along its execution, a datum $\langle$`test`$\rangle$ at node `test` appears; this is written $\overset{OK}{\Longrightarrow}$.

Definition 3.4 *$N \simeq' M$ if, for any observer K, it holds that $N \parallel K \overset{OK}{\Longrightarrow}$ if and only if $M \parallel K \overset{OK}{\Longrightarrow}$.*

Proposition 3.5 $\cong \ \subset \ \simeq \ = \ \simeq'$.

Proof: That $\cong$ is a sub-relation of $\simeq$ trivially follows from their definitions. The inclusion is strict because the latter equivalence abstracts from the branching structure of the equated nets, while the former one does not (because of reduction closure). This is standard in process calculi, e.g., in CCS and π-calculus.

We start proving that $\simeq \subseteq \simeq'$. Let $N \simeq M$ and pick up any observer K such that $N \parallel K \overset{OK}{\Longrightarrow}$. Then, by contextuality, $N \parallel K \simeq M \parallel K$ and, by barb preservation, $N \parallel K \Downarrow$ `test` (that comes from $N \parallel K \overset{OK}{\Longrightarrow}$) implies that $M \parallel K \Downarrow$ `test`. Since `test` is a name occuring only in K (by definition of observers), it must be $M \parallel K \overset{OK}{\Longrightarrow}$, as required.

Viceversa, we need to prove that $\simeq'$ is barb preserving and context closed. Let $N \simeq' M$.

Barb preservation. Let $N \Downarrow l$ and consider $K \triangleq$ `test` :: **in**$(!x)@l$.**out**(`test`)@`test`. Then, $N \parallel K \overset{OK}{\Longrightarrow}$ that, by hypothesis, implies $M \parallel K \overset{OK}{\Longrightarrow}$. Now, because of freshness of `test`, this is possible only if $M \Downarrow l$.

Context closure. The proof is by induction on the structure of the context $C[\cdot]$. The base case is trivial. For the inductive case, we have two possibilities:

- $C[\cdot] \triangleq \mathcal{D}[\cdot] \parallel H$. By induction, $\mathcal{D}[N] \simeq' \mathcal{D}[M]$; we pick up an observer K and prove that $C[N] \parallel K \overset{OK}{\Longrightarrow}$ implies $C[M] \parallel K \overset{OK}{\Longrightarrow}$ (by symmetry, this suffices). We now consider the observer $H \parallel K$; by Definition 3.4, by induction and by the fact that $\mathcal{D}[N] \parallel (H \parallel K) \overset{OK}{\Longrightarrow}$ we have that $\mathcal{D}[M] \parallel (H \parallel K) \overset{OK}{\Longrightarrow}$. The thesis easily follows by rule (PAss) and because $\equiv \subseteq \simeq'$.

- $C[\cdot] \triangleq (\nu l)\mathcal{D}[\cdot]$. By induction, $\mathcal{D}[N] \simeq' \mathcal{D}[M]$; we pick up an observer K and we prove that $C[N] \parallel K \overset{OK}{\Longrightarrow}$ implies $C[M] \parallel K \overset{OK}{\Longrightarrow}$. Since l is bound, we can assume, up-to alpha-equivalence, that $l \notin fn(K)$; in particular, $l \neq \texttt{test}$. Now, $C[N] \parallel K \overset{OK}{\Longrightarrow}$ if and only if $\mathcal{D}[N] \parallel K \overset{OK}{\Longrightarrow}$ (and similarly when replacing N with M). This suffices to conclude.
∎

The problem beyond barbed congruence and may testing is that context closure makes them hardly tractable, because of the universal quantification over all net contexts. In the following sections, we shall provide two tractable characterisations of these equivalences, as a *bisimulation* and as a *trace* equivalence.

Before doing this, we show that we can change observables without changing the congruences they induce; this proves the robustness of our touchstone equivalences and supports our choice. We shall give the explicit proof only for barbed congruence, but the same arguments hold also for may testing. Recalling from the Introduction, other two reasonable observables in a global computing framework can be existence of a specific (visible) datum in some node of a net, or existence of a specific (visible) datum in a specific node of a net.

Definition 3.6 (Alternative Reduction Barbed Congruences) *Let $\cong_1$ and $\cong_2$ be the reduction barbed congruences obtained by replacing the observable of Definition 3.1, respectively, with the following ones:*

1. *$N \downarrow \langle l \rangle$ iff $N \equiv (\nu\widetilde{l})(N' \parallel l' :: \langle l \rangle)$ for some $\widetilde{l}$ such that $\{l, l'\} \cap \widetilde{l} = \emptyset$*
2. *$N \downarrow_{l_1} \langle l_2 \rangle$ iff $N \equiv (\nu\widetilde{l})(N' \parallel l_1 :: \langle l_2 \rangle)$ for some $\widetilde{l}$ such that $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$*

We now prove that, thanks to contextuality, $\cong$, $\cong_1$ and $\cong_2$ do coincide.

Proposition 3.7 $\cong_1 = \cong_2 = \cong$.

Proof: Notice that we only need to consider barb preservation. Indeed, reduction closure and contextuality are ensured by definition by all the reduction barbed congruences we are considering.

$\cong_2 \subseteq \cong_1$**.** Let $N \cong_2 M$. Suppose that $N \Downarrow \langle l \rangle$. This implies that $\exists\, l' : N \Downarrow_{l'} \langle l \rangle$. Hence, by hypothesis, $M \Downarrow_{l'} \langle l \rangle$ that, by definition, implies $M \Downarrow \langle l \rangle$.

$\cong_1 \subseteq \cong$**.** Let $N \cong_1 M$ and $N \Downarrow l$. Then $M \Downarrow l$, otherwise the context $[\cdot] \parallel l' :: \textbf{in}(!x)@l.\textbf{out}(l')@l'$ (for l' fresh) would break $\cong_1$.

$\cong \subseteq \cong_2$**.** Let $N \cong M$ and $N \Downarrow_{l_1} \langle l_2 \rangle$. This means that $N \Rightarrow (\nu\widetilde{l})(N' \parallel l_1 :: \langle l_2 \rangle)$ for $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$. It must hold that $M \Downarrow_{l_1} \langle l_2 \rangle$ otherwise the context $[\cdot] \parallel l' :: \textbf{in}(l_2)@l_1.\textbf{out}()@l'$ (for l' fresh) would distinguish N and M (according to $\cong$).
∎

4 Bisimulation Equivalence

To coinductively capture barbed congruence, we introduce a labeled transition system (LTS) to make apparent the action a net is willing to perform in order to evolve. For the sake of presentation, we introduce the syntactic category of *inert components*

$$I ::= \textbf{nil} \quad \big| \quad \langle l \rangle$$

(LTS-Out) $l :: \mathbf{out}(l_2)@l_1.P \xrightarrow{\triangleright\, l_1} l :: P \parallel l_1 :: \langle l_2 \rangle$	(LTS-Eval) $l :: \mathbf{eval}(Q)@l_1.P \xrightarrow{\triangleright\, l_1} l :: P \parallel l_1 :: Q$
(LTS-In) $l :: \mathbf{in}(!\,x)@l_1.P \xrightarrow{l_2 \,\triangleleft\, l_1} l :: P[^{l_2}/_{x}] \parallel l_1 :: \mathbf{nil}$	(LTS-Match) $l :: \mathbf{in}(l_2)@l_1.P \xrightarrow{l_2 \,\triangleleft\, l_1} l :: P \parallel l_1 :: \mathbf{nil}$
(LTS-New) $l :: \mathbf{new}(l').P \xrightarrow{\tau} (\nu l')(l :: P \parallel l' :: \mathbf{nil})$	(LTS-Exists) $l :: I \xrightarrow{I\,@\,l} l :: \mathbf{nil}$
(LTS-Send) $\dfrac{N_1 \xrightarrow{\triangleright\, l} N_1' \qquad N_2 \xrightarrow{\mathbf{nil}\,@\,l} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$	(LTS-Comm) $\dfrac{N_1 \xrightarrow{l_2 \,\triangleleft\, l_1} N_1' \qquad N_2 \xrightarrow{\langle l_2 \rangle\,@\,l_1} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$
(LTS-Res) $\dfrac{N \xrightarrow{\alpha} N' \qquad l \notin n(\alpha)}{(\nu l)N \xrightarrow{\alpha} (\nu l)N'}$	(LTS-Open) $\dfrac{N \xrightarrow{\langle l' \rangle\,@\,l} N' \quad l' \neq l}{(\nu l')N \xrightarrow{(\nu l')\,\langle l' \rangle\,@\,l} N'}$
(LTS-Par) $\dfrac{N_1 \xrightarrow{\alpha} N_2 \qquad bn(\alpha) \cap fn(N) = \emptyset}{N_1 \parallel N \xrightarrow{\alpha} N_2 \parallel N}$	(LTS-Struct) $\dfrac{N \equiv N_1 \quad N_1 \xrightarrow{\alpha} N_2 \quad N_2 \equiv N'}{N \xrightarrow{\alpha} N'}$

Table 5: A Labelled Transition System

for grouping those components that are unable to perform any basic operation. The *labelled transition relation*, $\xrightarrow{\alpha}$, is defined as the least relation over nets induced by the inference rules in Table 5. Transition labels take the form

$$\chi \;::=\; \tau \;\;\big|\;\; (\nu\widetilde{l})\; I\,@\,l \qquad\qquad \alpha \;::=\; \chi \;\;\big|\;\; \triangleright\, l_1 \;\;\big|\;\; l_2 \,\triangleleft\, l_1$$

We will write $bn(\alpha)$ for $\widetilde{l}$ if $\alpha = (\nu\widetilde{l})\; I\,@\,l$ and for $\emptyset$, otherwise; $fn(\alpha)$ is defined accordingly. Moreover, from the definition of the LTS, it easily follows that, when α is of the form $(\nu\widetilde{l})\;\langle l' \rangle\,@\,l$, we can have either $\widetilde{l} = \emptyset$ (hence l' is not bound in the net executing α), or $\widetilde{l} = \{l'\}$.

Let us now briefly comment on some rules of the LTS; most of them are adapted from the π-calculus [38]. Rule (LTS-Exists) signals existence of nodes (label $\mathbf{nil}\,@\,l$) or of data (label $\langle l_2 \rangle\,@\,l_1$). Rules (LTS-Out) and (LTS-Eval) express the intention of spawning a component and require the existence of the target node to complete successfully (rule (LTS-Send)). Similarly, rules (LTS-In) (given in an early style) and (LTS-Match) express the intention of performing an input; this input is actually performed (rule (LTS-Comm)) only if the chosen datum is present in the target node. Notice that, in the right hand side of rules (LTS-In) and (LTS-Match), existence of the node target of the **in** can be assumed: indeed, if l_1 provides datum $\langle l_2 \rangle$, this implies that l_1 does exist. Rule (LTS-Open) signals extrusion of bound names; as in some presentation of the π-calculus, this rule is used to investigate the capability of processes to export bound names, rather than to extend the scope of bound names. To this last aim, law (Ext) is used; in fact, in rule (LTS-Comm) labels do not carry any restriction on names, whose scope must have been previously extended. Rules (LTS-Res), (LTS-Par) and (LTS-Struct) are standard. The structural congruence $\equiv$ involved in rule (LTS-Struct) is the one in Section 2.

Notation 4.1 We shall write $N \xrightarrow{\alpha}$ to mean that there exists a net N' such that $N \xrightarrow{\alpha} N'$. Alternatively, we could say that N can perform a α-step. Moreover, we shall usually denote relation composition by juxtaposition; thus, e.g., $N \xrightarrow{\alpha\ \alpha'} M$ means that there exists a net N' such that $N \xrightarrow{\alpha} N' \xrightarrow{\alpha'} M$. We shall use the convention that putting a bar over a relation means that such a relation does not hold (e.g. $N \overset{\alpha}{\not\to} N'$ means that N cannot reduce to N' performing α). As usual, we let $\Rightarrow$ to stand for $\xrightarrow{\tau}{}^*$, $\overset{\alpha}{\Longrightarrow}$ to stand for $\Rightarrow \xrightarrow{\alpha} \Rightarrow$, and $\overset{\hat{\alpha}}{\Longrightarrow}$ to stand for $\Rightarrow$, if $\alpha = \tau$, and for $\overset{\alpha}{\Longrightarrow}$, otherwise.

The LTS we have just defined is 'correct' w.r.t. the operational semantics of cKlaim, as stated by the following Proposition. Notice that $\longmapsto$ is the actual semantics of cKlaim; the LTS of Table 5 can be thought of as a technical device deployed to give a tractable formulation of barbed congruence.

Proposition 4.2 *$N \longmapsto M$ if and only if $N \xrightarrow{\tau} M$.*

Proof: Both the directions are proved by an easy induction on the inference of the judgements. ∎

Now, we prove some relationships between transitions of the LTS and the syntactical form of the net performing them.

Proposition 4.3 *The following facts hold:*

1. *$N \xrightarrow{\mathbf{nil}\ @\ l} N'$ if and only if $N \equiv N'' \parallel l :: \mathbf{nil}$; moreover, $N'' \equiv N' \equiv N$.*
2. *$N \xrightarrow{\langle l'\rangle\ @\ l} N'$ if and only if $N \equiv N'' \parallel l :: \langle l'\rangle$; moreover, $N' \equiv N'' \parallel l :: \mathbf{nil}$.*
3. *$N \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} N'$ if and only if $N \equiv (\nu l')(N'' \parallel l :: \langle l'\rangle)$ for $l \neq l'$; moreover, $N' \equiv N'' \parallel l :: \mathbf{nil}$.*

Proof: In all statements, the 'if' part is straightforward, by using (LTS-Exists) and (LTS-Struct)/(LTS-Par)/(LTS-Open)/(LTS-Res). For the converse, we explicitly consider the case $N \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} N'$ (the other cases are similar). The proof now proceeds by induction on the length of the inference of this reduction.

Base: the derivation is inferred using just two rules; such rules can only be (LTS-Exists) and (LTS-Open). Thus, $N \triangleq (\nu l')(l :: \langle l'\rangle)$, for $l' \neq l$, and $l :: \langle l'\rangle \xrightarrow{\langle l'\rangle\ @\ l} l :: \mathbf{nil}$. We trivially conclude.

Induction: we reason by case analysis on the last rule applied:

(LTS-Res) In this case, $N \triangleq (\nu l'')N_1$ and $l'' \notin \{l, l'\}$. Hence, $N_1 \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} N_1'$; by induction, $N_1 \equiv (\nu l')(N_2 \parallel l :: \langle l'\rangle)$ and $N_1' \equiv N_2 \parallel l :: \mathbf{nil}$ for $l \neq l'$. Hence, $N \equiv (\nu l'', l')(N_2 \parallel l :: \langle l'\rangle) \equiv (\nu l')((\nu l'')N_2 \parallel l :: \langle l'\rangle)$ and $N' \equiv (\nu l'')N_2 \parallel l :: \mathbf{nil}$.

(LTS-Par) In this case, $N \triangleq N_1 \parallel N_2$, $N_1 \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} N_1'$ and $l' \notin fn(N_2)$. The thesis easily follows from the inductive hypothesis.

(LTS-Struct) In this case, $N \equiv N_1$, $N_1 \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} N_1'$ and $N_1' \equiv N'$. The thesis follows by induction and by transitivity of $\equiv$. ∎

We now characterize barbed congruence by using the labels of the LTS instead of the universal quantification over all contexts. In this way, we obtain an alternative characterization of $\cong$ in terms of a labelled *bisimilarity*.

Definition 4.4 (Bisimilarity) *A symmetric relation $\mathfrak{R}$ between* cKlaim *nets is a* (weak) bisimulation *if for each $N\ \mathfrak{R}\ M$ it holds that:*

1. *if $N \xrightarrow{\chi} N'$ then, for some M', $M \overset{\hat{\chi}}{\Longrightarrow} M'$ and $N' \mathfrak{R} M'$;*
2. *if $N \xrightarrow{\triangleright\, l} N'$ then, for some M', $M \parallel l :: \textbf{nil} \Rightarrow M'$ and $N' \mathfrak{R} M'$;*
3. *if $N \xrightarrow{l_2 \,\triangleleft\, l_1} N'$ then, for some M', $M \parallel l_1 :: \langle l_2 \rangle \Rightarrow M'$ and $N' \mathfrak{R} M'$.*

Bisimilarity, $\approx$, *is the largest bisimulation.*

Our bisimulation is somehow inspired by that in [36]. The key idea is that, since sending operations are asynchronous, the evolution $N \xrightarrow{\triangleright\, l} N'$ can be simulated by a net M (in a context where locality l is present) through execution of some internal actions that lead to M'. Indeed, since we want our bisimulation to be a congruence, a context that provides the target locality of the sending action must not tell apart N and M. Hence, for $N \parallel l :: \textbf{nil}$ to be simulable by $M \parallel l :: \textbf{nil}$, it must hold that, upon transitions, N' be simulable by M'. Similar considerations hold also for input actions (third item of Definition 4.4), but the context now is $[\cdot] \parallel l_1 :: \langle l_2 \rangle$.

The LTS we developed does not use labels containing processes. Thus, the bisimulation we have just defined is clearly tractable and we strongly conjecture that it is decidable, under proper assumptions: techniques similar to those in [35] could be used here.

4.1 Soundness w.r.t. Barbed Congruence

The key result of this subsection is Lemma 4.7 that will easily allow us to conclude that bisimilarity is a sound proof technique for barbed congruence. To prove this result, we need some technical tools. First of all, we introduce the notion of *bisimulation up-to structural congruence*: it is defined as a labelled bisimulation except for the fact that the $\mathfrak{R}$ in the consequents of Definition 4.4 is replaced by the relation $\equiv \mathfrak{R} \equiv$. Lemma 4.5 shows that a bisimulation up-to $\equiv$ is a bisimulation. Then, Lemma 4.6 characterizes all the possible executions of the net $C[N]$ in terms of the evolutions of N and $C[\cdot]$ separately.

Lemma 4.5 *If $N \approx M$ then for any nets N' and M' such that $N \equiv N'$ and $M \equiv M'$ it holds that $N' \approx M'$.*

Proof: Let $\mathfrak{R} \triangleq \{ (N_1, N_2) \;:\; N_i \equiv N_i'$ *and* $N_1' \approx N_2' \}$. We shall prove that $\mathfrak{R}$ is a labelled bisimulation. Let $N_1 \xrightarrow{\alpha} M_1$; by (LTS-Struct) we have that $N_1' \equiv N_1 \xrightarrow{\alpha} M_1$. We only consider the case for $\alpha = \chi$; the other cases are similar. By hypothesis, $N_2' \xrightarrow{\hat{\chi}} M_2$ for some M_2 such that $M_1 \approx M_2$. Then, $N_2 \equiv N_2' \overset{\hat{\chi}}{\Longrightarrow} M_2$ and $(M_1, M_2) \in \mathfrak{R}$ because of reflexivity of $\equiv$. ∎

Lemma 4.6 *$C[N] \xrightarrow{\alpha} \overline{N}$ if and only if one of the following conditions hold:*

1. *$N \xrightarrow{\alpha} N'$ with $n(\alpha) \cap bn(C[\cdot]) = \emptyset$, or*
2. *$C[\mathbf{0}] \xrightarrow{\alpha} C'[\mathbf{0}]$, or*
3. *$N \xrightarrow{\alpha'} N'$ with $\alpha = (\nu l)\alpha'$, $C[\cdot] \triangleq C_1[(\nu l)C_2[\cdot]]$ and $fn(\alpha) \cap bn(C_1[\cdot], C_2[\cdot]) = \emptyset$, or*
4. *$C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{\textbf{nil}\ @\ l} H'$, $N \xrightarrow{\triangleright\, l} N'$ and $l \notin bn(C_2[\cdot])$, or*
5. *$C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{\triangleright\, l} H'$, $N \xrightarrow{\textbf{nil}\ @\ l} N'$ and $l \notin bn(C_2[\cdot])$, or*
6. *$C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{(\nu \tilde{l})\ \langle l_2 \rangle\ @\ l_1} H'$, $N \xrightarrow{l_2 \,\triangleleft\, l_1} N'$ and $\{l_1, l_2\} \cap bn(C_2[\cdot]) = \emptyset$, or*

7. *$C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{l_2 \blacktriangleleft l_1} H'$, $N \xrightarrow{(\nu\widetilde{l})\,\langle l_2\rangle\,@\,l_1} N'$ and $l_1 \notin bn(C_2[\cdot])$*

Moreover, the resulting net $\bar{N}$ is, respectively, structurally equivalent to $C[N']$, or $C'[N]$, or $C_1[C_2[N']]$, or $C[N']$, or $C_1[C_2[N] \parallel H']$, or $C_1[(\nu\widetilde{l})C_2[N' \parallel H']]$ (cases 6 and 7.). Finally, $\alpha = \tau$ in cases 4., 5., 6., and 7. .

Proof: The "if" part is trivial, by using the LTS of Table 5 and by observing that $M \xrightarrow{\alpha} M'$ with $n(\alpha) \cap bn(\mathcal{D}[\cdot]) = \emptyset$ implies $\mathcal{D}[M] \xrightarrow{\alpha} \mathcal{D}[M']$. The "only if" part is proved by induction on the length of the inference of $\xrightarrow{\alpha}$. In the base case (length 1), it must be $C[\cdot] \triangleq [\cdot]$; hence, obviously $C[N] \triangleq N \xrightarrow{\alpha} N' \triangleq C[N']$ (and hence we fall in case 1. of this Lemma). For the inductive step, we reason by case analysis on the last rule applied in the inference:

(LTS-Res**).** In this case, it must be

$$\frac{\mathcal{D}[N] \xrightarrow{\alpha} \bar{N}' \qquad l \notin n(\alpha)}{C[N] \triangleq (\nu l)\mathcal{D}[N] \xrightarrow{\alpha} (\nu l)\bar{N}'}$$

We can now apply induction and reason by analysis on the used case of this lemma.

1. $N \xrightarrow{\alpha} N'$, $n(\alpha) \cap bn(\mathcal{D}[\cdot]) = \emptyset$ and $\bar{N}' \equiv \mathcal{D}[N']$. Hence, we still fall in case 1. by using the context $C[\cdot] \triangleq (\nu l)\mathcal{D}[\cdot]$.
2. $\mathcal{D}[\mathbf{0}] \xrightarrow{\alpha} \mathcal{D}'[\mathbf{0}]$ and $\bar{N}' \equiv \mathcal{D}'[N]$. Hence, we still fall in case 2. with contexts $C[\cdot] \triangleq (\nu l)\mathcal{D}[\cdot]$ and $C'[\cdot] \triangleq (\nu l)\mathcal{D}'[\cdot]$.
3. $N \xrightarrow{\alpha'} N'$, $\alpha \triangleq (\nu l')\alpha'$, $\mathcal{D}[\cdot] \triangleq \mathcal{D}_1[(\nu l')\mathcal{D}_2[\cdot]]$ and $n(\alpha) \cap bn(\mathcal{D}_1[\cdot], \mathcal{D}_2[\cdot]) = \emptyset$; moreover, $\bar{N}' \equiv \mathcal{D}_1[\mathcal{D}_2[N']]$. Hence, we still fall in case 3. by using the contexts $C_1[\cdot] \triangleq (\nu l)\mathcal{D}_1[\cdot]$ and $C_2[\cdot] \triangleq \mathcal{D}_2[\cdot]$.
4. $\mathcal{D}[\cdot] \triangleq \mathcal{D}_1[\mathcal{D}_2[\cdot] \parallel H]$ with $H \xrightarrow{\mathbf{nil}\,@\,l} H'$, $N \xrightarrow{\triangleright\, l} N'$ and $l \notin bn(\mathcal{D}_2[\cdot])$; moreover, $\mathcal{D}[N] \xrightarrow{\tau} \mathcal{D}[N']$. Hence we still fall in case 4. by using the contexts $C_1[\cdot] \triangleq (\nu l)\mathcal{D}_1[\cdot]$ and $C_2[\cdot] \triangleq \mathcal{D}_2[\cdot]$.
5. 6. and 7. are similar.

(LTS-Open**).** In this case, it must be

$$\frac{\mathcal{D}[N] \xrightarrow{\langle l_2\rangle\,@\,l_1} \bar{N} \qquad l_2 \neq l_1}{C[N] \triangleq (\nu l_2)\mathcal{D}[N] \xrightarrow{(\nu l_2)\,\langle l_2\rangle\,@\,l_1} \bar{N}}$$

We can now apply induction and reason by analysis on the used case of this lemma; we have to consider only the first two cases.

1. $N \xrightarrow{\langle l_2\rangle\,@\,l_1} N'$, $\{l_1, l_2\} \cap bn(\mathcal{D}[\cdot]) = \emptyset$ and $\bar{N} \triangleq \mathcal{D}[N']$. Hence, we fall in case 3. by using the contexts $C_1[\cdot] \triangleq [\cdot]$ and $C_2[\cdot] \triangleq \mathcal{D}[\cdot]$.
2. $\mathcal{D}[\mathbf{0}] \xrightarrow{\langle l_2\rangle\,@\,l_1} \mathcal{D}'[\mathbf{0}]$ and $\bar{N} \triangleq \mathcal{D}'[N]$. Hence, we still fall in case 2. with contexts $C[\cdot] \triangleq (\nu l_2)\mathcal{D}[\cdot]$ and $C'[\cdot] \triangleq \mathcal{D}'[\cdot]$.

(LTS-Par**).** In this case, one of the following inferences should hold:

$$\frac{K \xrightarrow{\alpha} K' \quad bn(\alpha) \cap n(\mathcal{D}[N]) = \emptyset}{C[N] \triangleq \mathcal{D}[N] \parallel K \xrightarrow{\alpha} \mathcal{D}[N] \parallel K'} \qquad \text{or} \qquad \frac{\mathcal{D}[N] \xrightarrow{\alpha} \bar{N}' \quad bn(\alpha) \cap n(K) = \emptyset}{C[N] \triangleq \mathcal{D}[N] \parallel K \xrightarrow{\alpha} \bar{N}' \parallel K}$$

By using the first inference, we fall in case 2. with resulting context $C'[\cdot] \triangleq \mathcal{D}[\cdot] \parallel K'$. By using the second inference, we can apply inductive arguments similar to those used in the (LTS-Res) case, but now the context $C[\cdot]$ we consider is $\mathcal{D}[\cdot] \parallel K$ instead of $(\nu l)\mathcal{D}[\cdot]$.

(LTS-Send). In this case, one of the following inferences should hold:

$$\frac{K \xrightarrow{\triangleright\, l} K' \quad \mathcal{D}[N] \xrightarrow{\mathbf{nil}\ @\ l} \mathcal{D}[N]}{C[N] \triangleq \mathcal{D}[N] \parallel K \xrightarrow{\tau} \mathcal{D}[N] \parallel K'} \quad \text{or} \quad \frac{K \xrightarrow{\mathbf{nil}\ @\ l} K \quad \mathcal{D}[N] \xrightarrow{\triangleright\, l} \overline{N'}}{C[N] \triangleq \mathcal{D}[N] \parallel K \xrightarrow{\tau} \overline{N'} \parallel K}$$

To apply induction, notice that we only have to consider the first two cases of this lemma, since the actions fired by $\mathcal{D}[N]$ are different from τ and have no restricted names. For the first reduction, we have the following cases.

1. $N \xrightarrow{\mathbf{nil}\ @\ l} N$ and $l \notin bn(\mathcal{D}[\cdot])$. Hence, we fall in case 5. by using $C_1[\cdot] \triangleq [\cdot]$, $C_2[\cdot] \triangleq \mathcal{D}[\cdot]$ and $H \triangleq K$.
2. $\mathcal{D}[\mathbf{0}] \xrightarrow{\mathbf{nil}\ @\ l} \mathcal{D}[\mathbf{0}]$; hence, we still fall in case 2. with resulting context $C'[\cdot] \triangleq \mathcal{D}[\cdot] \parallel K' \parallel l :: C$.

For the second reduction, we have similar cases; we just list the differences.

1. $\overline{N'} \triangleq \mathcal{D}[N']$ and we fall into case 4. .
2. $\overline{N'} \triangleq \mathcal{D}'[N]$ and the resulting context is $C'[\cdot] \triangleq \mathcal{D}'[\cdot] \parallel K$.

(LTS-Comm). In this case, one of the following inferences should hold:

$$\frac{\mathcal{D}[N] \xrightarrow{l_2 \,\triangleleft\, l_1} \overline{N'} \quad K \xrightarrow{\langle l_2 \rangle\ @\ l_1} K'}{C[N] \triangleq \mathcal{D}[N] \parallel K \xrightarrow{\tau} \overline{N'} \parallel K'} \quad \text{or} \quad \frac{K \xrightarrow{l_2 \,\triangleleft\, l_1} K' \quad \mathcal{D}[N] \xrightarrow{\langle l_2 \rangle\ @\ l_1} \overline{N'}}{C[N] \triangleq \mathcal{D}[N] \parallel K \xrightarrow{\tau} \overline{N'} \parallel K'}$$

Like before, we just have two possible inductive cases for each reduction (namely, the first two of this lemma). For the first reduction we have the following cases.

1. $N \xrightarrow{l_2 \,\triangleleft\, l_1} N'$, $\{l_1, l_2\} \cap bn(\mathcal{D}[\cdot]) = \emptyset$ and $\overline{N'} \triangleq \mathcal{D}[N']$. Hence, we fall in case 6. by using $C_1[\cdot] \triangleq [\cdot]$, $C_2[\cdot] \triangleq \mathcal{D}[\cdot]$ and $H \triangleq K$.
2. $\mathcal{D}[\mathbf{0}] \xrightarrow{l_2 \,\triangleleft\, l_1} \mathcal{D}'[\mathbf{0}]$ and $\overline{N'} \triangleq \mathcal{D}'[N]$. Hence, we still fall in case 2. with resulting context $C'[\cdot] \triangleq \mathcal{D}'[\cdot] \parallel K'$.

For the second reduction, we have similar cases. The only difference is that case 1. in the induction leads to case 7. in the conclusion.

(LTS-Struct). In this case, it must be

$$\frac{C[N] \equiv M_1 \xrightarrow{\alpha} M_2 \equiv \overline{N}}{C[N] \xrightarrow{\alpha} \overline{N}}$$

We now proceed by induction on the structure of context $C[\cdot]$. The base case (for $C[\cdot] \triangleq [\cdot]$) trivially falls in case 1. of this Lemma. For the inductive case, let us reason by case analysis on according to the structure of $C[\cdot]$:

$C[\cdot] \triangleq (\nu l)\mathcal{D}[\cdot]$. We furtherly identify three sub-cases:

- if $M_1 \triangleq (\nu l)M$ and $l \in bn(\alpha)$, for some $M \equiv \mathcal{D}[N]$, then we can apply the structural induction to $\mathcal{D}[N] \xrightarrow{\alpha'} M'$, for some $M' \equiv M_2$ and $\alpha = (\nu l)\alpha'$, and fall in one of the first two cases of this Lemma. By using rule (LTS-Open), we can conclude that $\mathcal{C}[N] \xrightarrow{\alpha} \overline{N}$ falls in cases 2. or 3. of this Lemma.
- if $M_1 \triangleq (\nu l)M$ and $l \notin bn(\alpha)$, for some $M \equiv \mathcal{D}[N]$, then we can apply the structural induction to $\mathcal{D}[N] \xrightarrow{\alpha} M'$, for some M' such that $M_2 \equiv (\nu l)M'$, falling in one of the cases of this Lemma. Then, by using (LTS-Res), we can conclude that $\mathcal{C}[N] \xrightarrow{\alpha} \overline{N}$ falls in the same case of this Lemma.
- otherwise, we can prove that $\mathcal{C}[N] \equiv M_1' \xrightarrow{\alpha} M_2$ such that $M_1' \triangleq (\nu l)M$ by using a no longer inference (but possibly using more structural laws). Hence, we can reduce this case to the previous one.

$\mathcal{C}[\cdot] \triangleq \mathcal{D}[\cdot] \parallel K$. Because of the structure of $\mathcal{C}[\cdot]$, it can be one of the following cases:

- $K \xrightarrow{\alpha} K'$ and $\overline{N} \equiv \mathcal{D}[N] \parallel K'$. In this case, we are trivially in case 2. of this Lemma.
- $\mathcal{D}[N] \xrightarrow{\alpha} \overline{N}'$ and $\overline{N} \equiv \overline{N}' \parallel K$. In this case, we use the structural induction.
- If $\alpha = \tau$ then other four cases are possible:
  - $\mathcal{D}[N] \xrightarrow{\triangleright\, l} \overline{N}', K \xrightarrow{\mathbf{nil}\ @\ l} K$ and $\overline{N} \equiv \overline{N}' \parallel K$. By structural induction, it can be that either $N \xrightarrow{\triangleright\, l} N'$, or $\mathcal{D}[\mathbf{0}] \xrightarrow{\triangleright\, l} \mathcal{D}'[\mathbf{0}]$. In both cases is easy to conclude.
  - $\mathcal{D}[N] \xrightarrow{(\nu\tilde{l})\ \langle l_2\rangle\ @\ l_1} \overline{N}', K \xrightarrow{l_2\ \triangleleft\ l_1} K'$ and $\overline{N} \equiv (\nu\tilde{l})(\overline{N}' \parallel K')$. This case is similar to the previous one.
  - $\mathcal{D}[N] \xrightarrow{\mathbf{nil}\ @\ l} \mathcal{D}[N], K \xrightarrow{\triangleright\, l} K'$ and $\overline{N} \equiv \mathcal{D}[N] \parallel K'$. By structural induction, it can be one of the first two cases of this Lemma and we can easily conclude.
  - $\mathcal{D}[N] \xrightarrow{l_2\ \triangleleft\ l_1} \overline{N}', K \xrightarrow{(\nu\tilde{l})\ \langle l_2\rangle\ @\ l_1} K'$ and $\overline{N} \equiv (\nu\tilde{l})(\overline{N}' \parallel K')$. This case is similar to the previous one. ∎

Lemma 4.7 *$\approx$ is a congruence relation.*

Proof:

We start by proving that $\approx$ is substitutive w.r.t. to the net contexts $\mathcal{C}[\cdot]$, namely that $N \approx M$ implies $\mathcal{C}[N] \approx \mathcal{C}[M]$ for each $\mathcal{C}[\cdot]$. To this aim, we prove that

$$\mathfrak{R} \triangleq \{\ (\mathcal{C}[N], \mathcal{C}[M]) : N \approx M\ \}$$

is a bisimulation up-to $\equiv$. Let $\mathcal{C}[N] \xrightarrow{\alpha} \overline{N}$; according to Lemma 4.6 we have to examine seven cases.

1. $N \xrightarrow{\alpha} N'$ for $n(\alpha) \cap bn(\mathcal{C}[\cdot]) = \emptyset$; we reason by case analysis on α.

   $\alpha = \chi$. By hypothesis of bisimilarity, $M \overset{\hat{\chi}}{\Longrightarrow} M'$ and $N' \approx M'$. Hence, trivially, $\mathcal{C}[M] \overset{\hat{\chi}}{\Longrightarrow} \mathcal{C}[M']$ and, by definition of $\mathfrak{R}$, it holds that $\mathcal{C}[N']\ \mathfrak{R}\ \mathcal{C}[M']$.

   $\alpha = \triangleright\, l$. By hypothesis of bisimilarity, $M \parallel l :: \mathbf{nil} \Rightarrow M'$ and $N' \approx M'$. Since $l \notin bn(\mathcal{C}[\cdot])$, we have that $\mathcal{C}[M] \parallel l :: \mathbf{nil} \equiv \mathcal{C}[M \parallel l :: \mathbf{nil}] \Rightarrow \mathcal{C}[M']$ and $\mathcal{C}[N']\ \mathfrak{R}\ \mathcal{C}[M']$ up-to structural equivalence.

   $\alpha = l_2 \triangleleft l_1$. By hypothesis, $\{l_1, l_2\} \cap bn(\mathcal{C}[\cdot]) = \emptyset$; thus, $\mathcal{C}[M] \parallel l_1 :: \langle l_2\rangle \equiv \mathcal{C}[M \parallel l_1 :: \langle l_2\rangle]$. By hypothesis of bisimilarity, $M \parallel l_1 :: \langle l_2\rangle \Rightarrow M'$ and $N' \approx M'$. Thus, $\mathcal{C}[M] \parallel l_1 :: \langle l_2\rangle \Rightarrow \mathcal{C}[M']$ and $\mathcal{C}[N']\ \mathfrak{R}\ \mathcal{C}[M']$ up-to $\equiv$.

2. $C[\mathbf{0}] \xrightarrow{\alpha} C'[\mathbf{0}]$; trivially, $C[M] \xrightarrow{\alpha} C'[M]$. Moreover, by definition of $\mathfrak{R}$, we have that
   - if $\alpha = \chi$ then $C'[N] \;\mathfrak{R}\; C'[M]$.
   - if $\alpha = \;\triangleright l\;$, then $C[M] \parallel l :: \mathbf{nil} \Rightarrow C'[M]$ and $C'[N] \;\mathfrak{R}\; C'[M]$.
   - if $\alpha = \; l_2 \triangleleft l_1\;$, then $C[M] \parallel l :: \langle l_2 \rangle \Rightarrow C'[M]$ and $C'[N] \;\mathfrak{R}\; C'[M]$

3. $N \xrightarrow{\langle l_2 \rangle \,@\, l_1} N'$, $C[\cdot] \triangleq C_1[(\nu l_2)C_2[\cdot]]$, for $l_1 \notin bn(C_1[\cdot], C_2[\cdot])$, and $\overline{N} \triangleq C_1[C_2[N']]$. By hypothesis of bisimilarity, $M \xRightarrow{\langle l_2 \rangle \,@\, l_1} M'$ and $N' \approx M'$. Thus, $C[M] \xRightarrow{(\nu l_2)\,\langle l_2 \rangle \,@\, l_1} C_1[C_2[M']]$. The thesis easily follows.

4. By hypothesis, $C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{\mathbf{nil} \,@\, l} H'$, $N \xrightarrow{\triangleright l} N'$, for $l \notin bn(C_2[\cdot])$, and $\overline{N} \equiv C[N']$. By Proposition 4.3, $H \equiv H' \equiv H \parallel l :: \mathbf{nil}$; thus, $C[M] \equiv C[M \parallel l :: \mathbf{nil}]$. By hypothesis of bisimilarity, $M \parallel l :: \mathbf{nil} \Rightarrow M'$ and $N' \approx M'$. Hence, $C[M] \Rightarrow C[M']$ and $\overline{N} \;\mathfrak{R}\; M'$ up-to $\equiv$.

5. $C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{\triangleright l} H'$, $N \xrightarrow{\mathbf{nil} \,@\, l} N'$ and $l \notin bn(C_2[\cdot])$. By hypothesis of bisimilarity, $M \xRightarrow{\mathbf{nil} \,@\, l} M$ and $N' \approx M'$. Hence, $C[M] \Rightarrow C_1[C_2[M] \parallel H']$. The thesis easily follows.

6. $C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{(\nu\tilde{l})\,\langle l_2 \rangle \,@\, l_1} H'$, $N \xrightarrow{l_2 \triangleleft l_1} N'$, for $\{l_1, l_2\} \cap bn(C_2[\cdot]) = \emptyset$, and $\overline{N} \equiv C_1[(\nu\tilde{l})(C_2[N'] \parallel H')]$. By Proposition 4.3, $H \equiv (\nu\tilde{l})(H'' \parallel l_1 :: \langle l_2 \rangle)$ and $H' \equiv H'' \parallel l_1 :: \mathbf{nil}$; thus, $C[M] \equiv C_1[(\nu\tilde{l})(C_2[M \parallel l_1 :: \langle l_2 \rangle] \parallel H'')]$. By hypothesis of bisimilarity, $M \parallel l_1 :: \langle l_2 \rangle \Rightarrow M'$ and $N' \approx M'$. Hence $C[M] \Rightarrow C_1[(\nu\tilde{l})(C_2[M'] \parallel H'')]$ and $\overline{N} \;\mathfrak{R}\; C_1[(\nu\tilde{l})(C_2[M'] \parallel H')]$ up-to $\equiv$.

7. $C[\cdot] \triangleq C_1[C_2[\cdot] \parallel H]$ with $H \xrightarrow{l_2 \triangleleft l_1} H'$, $N \xrightarrow{(\nu\tilde{l})\,\langle l_2 \rangle \,@\, l_1} N'$, for $\{l_1, l_2\} \cap bn(C_2[\cdot]) = \emptyset$, and $\overline{N} \equiv C_1[(\nu\tilde{l})(C_2[N'] \parallel H')]$. By hypothesis of bisimilarity, $M \xRightarrow{(\nu\tilde{l})\,\langle l_2 \rangle \,@\, l_1} M'$ and $N' \approx M'$. Hence, $C[M] \Rightarrow C_1[(\nu\tilde{l})C_2[M' \parallel H']]$ and $\overline{N} \;\mathfrak{R}\; C_1[(\nu\tilde{l})(C_2[M'] \parallel H')]$ up-to $\equiv$.

We are left with proving that $\approx$ is an equivalence relation. Reflexivity and symmetry follow by definition. To prove transitivity, we consider the relation $\mathfrak{R} \triangleq \{\, (N_1, N_2) \;:\; N_1 \approx\approx N_2 \}$ and prove that it is a bisimulation. Let $N_1 \approx M \approx N_2$, $N_1 \xrightarrow{\alpha} N'_1$ and let us reason by case analysis on α:

$\alpha = \chi$. In this case, $M \xRightarrow{\hat{\chi}} M'$ for some M' such that $N'_1 \approx M'$. If $M' \equiv M$, then we conclude up-to $\equiv$. Otherwise, it must be that $N_2 \xRightarrow{\hat{\chi}} N'_2$ and $M' \approx N'_2$; hence $N'_1 \approx\approx N'_2$ and $N'_1 \;\mathfrak{R}\; N'_2$.

$\alpha = \;\triangleright l$. By hypothesis, $M \parallel l :: \mathbf{nil} \Rightarrow M'$ and $N'_1 \approx M'$. By context closure, we have that $M \parallel l :: \mathbf{nil} \approx N_2 \parallel l :: \mathbf{nil}$; hence, $N_2 \parallel l :: \mathbf{nil} \Rightarrow N'_2$ and $M' \approx N'_2$. It is easy to conclude that $N'_1 \;\mathfrak{R}\; N'_2$.

$\alpha = \; l_2 \triangleleft l_1$. Similarly, $M \parallel l_1 :: \langle l_2 \rangle \Rightarrow M'$ and $N'_1 \approx M'$. Moreover, $N_2 \parallel l_1 :: \langle l_2 \rangle \Rightarrow N'_2$ and $M' \approx N'_2$, that implies $N'_1 \;\mathfrak{R}\; N'_2$. ∎

Theorem 4.8 (Soundness of $\approx$ w.r.t. $\cong$) *If $N \approx M$ then $N \cong M$.*

Proof: We shall now prove that $\approx$ is barb preserving, reduction closed and contextual. By definition, this implies that $\approx \subseteq \cong$.

- If $N \Downarrow l$ then $N \xRightarrow{(\nu\widetilde{l})\,\langle l' \rangle\,@\,l}$, for some l' and $\widetilde{l}$ such that $l \notin \widetilde{l}$; hence, by hypothesis of bisimilarity, $M \xRightarrow{(\nu\widetilde{l})\,\langle l' \rangle\,@\,l}$ and thus $M \Downarrow l$ (these implications rely on Proposition 4.3 and Definition 4.4).

- By Proposition 4.2, $N \longmapsto N'$ implies that $N \xrightarrow{\tau} N'$; this, in turn, implies, by hypothesis of bisimilarity, that $M \Rightarrow M'$ (and, again, by Proposition 4.2 this means $M \models\!\!\Rightarrow M'$) and $N' \approx M'$.

- By Lemma 4.7, for all net context $C[\cdot]$, it holds that $C[N] \approx C[M]$. ∎

4.2 Completeness w.r.t. Barbed Congruence

We now want to prove the converse, namely that all barbed congruent processes are also bisimilar. To this aim, we need three technical results. The first one gives some simple equations that hold true w.r.t. barbed congruence. The second result gives an alternative characterization of the contextuality property of Definition 3.3. The third result states that we can throw away fresh localities hosting restricted data without breaking barbed congruence.

Proposition 4.9 *The following facts hold:*

1. $(\nu l')(\, l :: \mathbf{in}(!x)@l'.P \parallel l' :: \langle d \rangle\,) \cong (\nu l')(l :: P[^{d}/_{x}])$
2. $l :: \mathbf{out}(l'')@l'.P \parallel l' :: \mathbf{nil} \cong l :: P \parallel l' :: \langle l'' \rangle$
3. $l :: \mathbf{eval}(Q)@l'.P \parallel l' :: \mathbf{nil} \cong l :: P \parallel l' :: Q$
4. $(\nu l)N \cong N$ *whenever* $l \notin fn(N)$

Proof: The first three equations can be easily proved by providing a proper bisimulation containing each of them; this fact, together with Theorem 4.8, proves part (1)/(2)/(3). The last equation is proved by first observing that $\equiv\ \subseteq\ \cong$ (this can be easily proved). Then, $(\nu l)N \equiv (\nu l)(N \parallel l :: \mathbf{nil}) \equiv N \parallel (\nu l)(l :: \mathbf{nil}) \cong N \parallel \mathbf{0} \equiv N$ (indeed, $(\nu l)(l :: \mathbf{nil}) \cong \mathbf{0}$, as it can be easily verified). Thus, by inclusion of $\equiv$ in $\cong$ and by transitivity of $\cong$, the claim holds. ∎

Lemma 4.10 *A relation* $\Re$ *is contextual if and only if*

1. $N \Re M$ *implies that* $N \parallel l :: P \Re M \parallel l :: P$ *for any name* l *and process* P*, and*
2. $N \Re M$ *implies that* $(\nu l)N \Re (\nu l)M$ *for any name* l

Proof: It is trivial to prove that contextuality implies points (1) and (2) of this Lemma. For the converse, let us assume $N \Re M$ and pick up a context $C[\cdot]$. We now proceed by induction on the structure of $C[\cdot]$. The base case is trivial. For the inductive case, we have two possibilities:

1. $C[\cdot] \triangleq \mathcal{D}[\cdot] \parallel K$. By induction, we have that $\mathcal{D}[N] \Re \mathcal{D}[M]$. We now proceed by induction on the structure of K. The base case is trivial, up-to $\equiv$. For the inductive case, it can be either $K \triangleq l :: P \parallel K'$ or $K \triangleq (\nu l)K'$. In the first case, by point (1) of this Lemma, it holds that $\mathcal{D}[N] \parallel l :: P \Re \mathcal{D}[M] \parallel l :: P$; then, by second induction, we can conclude $C[N] \Re C[M]$. In the second case, we can always assume that l is fresh for $\mathcal{D}[\cdot]$, N and M (this is always possible, up-to alpha-equivalence). By second induction, we have that $\mathcal{D}[N] \parallel K' \Re \mathcal{D}[M] \parallel K'$ and, by point (2) of this Lemma, we can conclude up-to $\equiv$.

2. $C[\cdot] \triangleq (\nu l)\mathcal{D}[\cdot]$. By induction we have that $\mathcal{D}[N] \Re \mathcal{D}[M]$; thus, by point (2) of this Lemma, it holds that $(\nu l)\mathcal{D}[N] \Re (\nu l)\mathcal{D}[M]$, i.e. $C[N] \Re C[M]$. ∎

Lemma 4.11 *If* $(\nu l)(N \parallel l_f :: \langle l \rangle) \cong (\nu l)(M \parallel l_f :: \langle l \rangle)$ *and* l_f *is fresh for* N *and* M*, then* $N \cong M$.

Proof: It suffices to prove that

$$\Re \triangleq \{ (N, M) \ : \ l_f \notin n(N, M) \ \wedge \ (\nu l)(N \parallel l_f :: \langle l \rangle) \cong (\nu l)(M \parallel l_f :: \langle l \rangle) \}$$

is barb preserving, reduction closed and contextual. Let $N \ \Re \ M$.

Barb preservation. Let $N \Downarrow l'$ for $l' \neq l$. Then, it trivially holds that $(\nu l)(N \parallel l_f :: \langle l \rangle) \Downarrow l'$, $(\nu l)(M \parallel l_f :: \langle l \rangle) \Downarrow l'$ and $M \Downarrow l'$ (indeed, $l' \neq l_f$ because of freshness of l_f).

Now, let $N \Downarrow l$. The argument above does not hold because $(\nu l)(N \parallel l_f :: \langle l \rangle) \not\Downarrow l$. However, we can consider the context $C[\cdot] \triangleq [\cdot] \parallel l'_f :: \mathbf{in}(!x)@l_f.\mathbf{in}(!y)@x.\mathbf{out}()@l'_f$, where l'_f is fresh. Now, $C[(\nu l)(N \parallel l_f :: \langle l \rangle)] \Downarrow l'_f$; hence, by hypothesis, $C[(\nu l)(M \parallel l_f :: \langle l \rangle)] \Downarrow l'_f$. Because of freshness of l_f and l'_f, it must be that $M \Downarrow l$.

Reduction closure. Let $N \longmapsto N'$; thus, $(\nu l)(N \parallel l_f :: \langle l \rangle) \longmapsto (\nu l)(N' \parallel l_f :: \langle l \rangle)$. By hypothesis, this fact implies that $(\nu l)(M \parallel l_f :: \langle l \rangle) \models\Rightarrow \overline{M}$ such that $(\nu l)(N' \parallel l_f :: \langle l \rangle) \cong \overline{M}$. Since $l_f \notin n(M)$, $l_f :: \langle l \rangle$ is not involved in the transition; thus it follows that $M \models\Rightarrow M'$ and $\overline{M} \equiv (\nu l)(M' \parallel l_f :: \langle l \rangle)$. Thus, the claim is proved up-to $\equiv$.

Contextuality. According to Lemma 4.10, we have to prove just two cases.

1. *For any l' and P, it holds that $N \parallel l' :: P \ \Re \ M \parallel l' :: P$.*

   Let us first assume that $l \notin fn(l' :: P)$. In this case, it is easy to conclude because $(\nu l)(N \parallel l_f :: \langle l \rangle) \parallel l' :: P \equiv (\nu l)(N \parallel l' :: P \parallel l_f :: \langle l \rangle)$ (and similarly when replacing N with M), by transitivity of $\cong$ and because $\equiv \subseteq \cong$.

   We now consider the case in which $l \in fn(P)$ but $l' \neq l$. We use the context $C[\cdot] \triangleq (\nu l_f)([\cdot] \parallel l' :: \mathbf{in}(!x)@l_f.\mathbf{out}(x)@l'_f.P[^x/_l] \parallel l'_f :: \mathbf{nil})$, where l'_f is a fresh name. Then, $C[(\nu l)(N \parallel l_f :: \langle l \rangle)] \cong (\nu l_f, l)(N \parallel l' :: \mathbf{out}(l)@l'_f.P \parallel l'_f :: \mathbf{nil}) \cong (\nu l_f, l)(N \parallel l' :: P \parallel l'_f :: \langle l \rangle) \cong (\nu l)(N \parallel l' :: P \parallel l'_f :: \langle l \rangle)$. These equalities hold true by using, resp., Proposition 4.9(1), (2) and (4). Similarly, $C[(\nu l)(M \parallel l_f :: \langle l \rangle)] \cong (\nu l)(M \parallel l' :: P \parallel l'_f :: \langle l \rangle)$. By transitivity, $C[(\nu l)(N \parallel l_f :: \langle l \rangle)] \cong C[(\nu l)(M \parallel l_f :: \langle l \rangle)]$ implies that $(\nu l)(N \parallel l' :: P \parallel l'_f :: \langle l \rangle) \cong (\nu l)(M \parallel l' :: P \parallel l'_f :: \langle l \rangle)$ and thus $N \parallel l' :: P \ \Re \ M \parallel l' :: P$.

   The case for $l' = l$ is dealt with similarly. It uses context $C[\cdot] \triangleq (\nu l_f)([\cdot] \parallel l'_f :: \mathbf{in}(!x)@l_f.\mathbf{eval}(P[^x/_l])@x.\mathbf{out}(x)@l'_f)$, where l'_f is a fresh name, and Proposition 4.9(3).

2. *For any l', it holds that $(\nu l')N \ \Re \ (\nu l')M$.*

   Let $l' \neq l, l_f$. Then $(\nu l', l)(N \parallel l_f :: \langle l \rangle) \equiv (\nu l)((\nu l')N \parallel l_f :: \langle l \rangle)$ (and similarly when replacing N with M). By transitivity of $\cong$ and because $\equiv \subseteq \cong$ we can easily conclude.

   Let $l' = l \neq l_f$. Then we consider the context $C[\cdot] \triangleq (\nu l_f)([\cdot] \parallel l'_f :: \mathbf{in}(!x)@l_f.\mathbf{new}(l'').\mathbf{out}(l'')@l'_f)$, for l'_f and l'' fresh. Then $C[(\nu l)(N \parallel l_f :: \langle l \rangle)] \cong (\nu l'')((\nu l)N \parallel l'_f :: \langle l'' \rangle)$ (and similarly when replacing N with M). Thus, we can conclude that $(\nu l'')((\nu l)N \parallel l'_f :: \langle l'' \rangle) \cong (\nu l'')((\nu l)M \parallel l'_f :: \langle l'' \rangle)$ that implies $(\nu l)N \ \Re \ (\nu l)M$.

   Finally, let $l' = l_f \neq l$. Then we consider the context $C[\cdot] \triangleq (\nu l_f)([\cdot] \parallel l'_f :: \mathbf{in}(!x)@l_f.\mathbf{out}(x)@l'_f)$, for l'_f fresh. Then $C[(\nu l)(N \parallel l_f :: \langle l \rangle)] \cong (\nu l)((\nu l_f)N \parallel l'_f :: \langle l \rangle)$ (and similarly when replacing N with M). Thus, we can conclude. ∎

Theorem 4.12 (Completeness of $\approx$ w.r.t. $\cong$) *If $N \cong M$ then $N \approx M$.*

Proof: It is enough to prove that $\cong$ is a bisimulation. We now pick up a transition $N \xrightarrow{\alpha} N'$ and reason by case analysis on α.

$\alpha = \tau$. This case is simple because of reduction closure.

$\alpha = \langle l \rangle \ @\ l'$. In this case, we consider the context $C[\cdot] \triangleq [\cdot] \parallel l_f :: \mathbf{in}(l)@l'.\mathbf{new}(l'').\mathbf{out}(l'')@l_f$, for l_f and l'' fresh, and the reduction $C[N] \models\!\Rightarrow (\nu l'')(N' \parallel l_f :: \langle l'' \rangle)$. By contextuality and reduction closure, it must be that $C[M] \models\!\Rightarrow \bar{M}$ such that $(\nu l'')(N' \parallel l_f :: \langle l'' \rangle) \cong \bar{M}$. This implies that $\bar{M} \Downarrow l_f$; but this is possible only if $M \Rightarrow\xrightarrow{\langle l \rangle \ @\ l'} M_1 \Rightarrow M_2$. Moreover, because of freshness of l_f and l'', we can state that $\bar{M} \equiv (\nu l'')(M_2 \parallel l_f :: \langle l'' \rangle)$. Thus, $M \xRightarrow{\langle l \rangle \ @\ l'} M_2$ and, by Lemma 4.11, $N' \cong M_2$.

$\alpha = (\nu l)\ \langle l \rangle \ @\ l'$. We now consider the context $C[\cdot] \triangleq [\cdot] \parallel l_f :: \mathbf{in}(!x)@l'.\mathbf{out}(x)@l_f$, for l_f fresh, and the reduction $C[N] \models\!\Rightarrow (\nu l)(N' \parallel l_f :: \langle l \rangle)$. By arguments similar to those above, we are ensured that $M \xRightarrow{\_ \ @\ l'}$; we now prove that M (weakly) offers in l' a restricted locality. By contradiction, let us assume that M only offers in l' free names and let us pick up one of these names, say l''. Then $M \xRightarrow{\langle l'' \rangle \ @\ l'} M'$ and $C[M] \models\!\Rightarrow M' \parallel l_f :: \langle l'' \rangle$. But it cannot be that $(\nu l)(N' \parallel l_f :: \langle l \rangle) \cong M' \parallel l_f :: \langle l'' \rangle$ for any M' because of the context $[\cdot] \parallel l'_f :: \mathbf{in}(l'')@l_f.\mathbf{out}()@l'_f$. Hence, l'' must be bound in M; we can now alpha-convert l'' to l in M (this is possible since we are assuming that bound names are pairwise distinct and different from the free ones, and hence $l \notin n(M)$). Thus, $M \xRightarrow{(\nu l)\ \langle l \rangle \ @\ l'} M''$ and $(\nu l)(N' \parallel l_f :: \langle l \rangle) \cong (\nu l)(M'' \parallel l_f :: \langle l \rangle)$; by Lemma 4.11, we can conclude.

$\alpha = \mathbf{nil} \ @\ l$. We now consider the context $C[\cdot] \triangleq [\cdot] \parallel l_f :: \mathbf{eval}(\mathbf{nil})@l.\mathbf{new}(l').\ \mathbf{out}(l')@l_f$, for l_f fresh, and the reduction $C[N] \models\!\Rightarrow (\nu l')(N' \parallel l_f :: \langle l' \rangle)$. Like before, we have that $M \xRightarrow{\mathbf{nil} \ @\ l} M'$ and $(\nu l')(N' \parallel l_f :: \langle l' \rangle) \cong (\nu l')(M' \parallel l_f :: \langle l' \rangle)$ that suffices to conclude.

$\alpha = \triangleright\, l$. We consider the context $[\cdot] \parallel l :: \mathbf{nil}$ and the reduction $N \parallel l :: \mathbf{nil} \longmapsto N'$. Then, by contextuality and reduction closure, $M \parallel l :: \mathbf{nil} \models\!\Rightarrow M'$ and $N' \cong M'$. This suffices to conclude (see Definition 4.4.2).

$\alpha = l_2 \triangleleft l_1$. This case is similar to the previous one; we just consider the context $[\cdot] \parallel l_1 :: \langle l_2 \rangle$ and the reduction $N \parallel l_1 :: \langle l_2 \rangle \longmapsto N'$. ∎

Corollary 4.13 (Tractable Characterization of Barbed Congruence) $\approx\, = \,\cong$.

5 Trace Equivalence

In this section, we develop a tractable characterization of may testing. For some well-known process calculi, may testing coincides with trace equivalence [22, 8, 9]; in this section, we show how a similar result is obtained also in the setting of cKlaim. To the best of our knowledge, this is the first tractable characterization of may testing for a distributed language with process mobility.

The idea beyond trace equivalence is that N and M are related if and only if the sets of their traces coincide. Put in another form, if N exhibits a sequence of visible actions σ, then M must exhibit σ as well, and viceversa. In an asynchronous setting [9, 16], this requirement must be properly weakened, since the discriminating power of asynchronous contexts is weaker: in the asynchronous π-calculus, for example, contexts cannot observe input actions.

To define a proper trace equivalence we slightly modify the LTS of Table 5 by adding the rule

$$\text{(LTS-Rcv)} \qquad \frac{N \xrightarrow{l_2 \triangleleft l_1} N' \qquad l_2 \notin fn(N)}{N \xrightarrow{(\nu l_2)\ l_2 \triangleleft l_1} N' \parallel l_2 :: \mathbf{nil}}$$

(L1)	$\sigma \cdot (\nu\widetilde{l})\sigma'$	$\leq_0$	$\sigma \cdot (\nu\widetilde{l}) \bullet \vartriangle l \cdot \sigma'$	if $(\nu\widetilde{l})\sigma' \neq \bot$
(L2)	$\sigma \cdot (\nu\widetilde{l})(\phi \cdot \bullet\vartriangle l \cdot \sigma')$	$\leq_0$	$\sigma \cdot (\nu\widetilde{l}) \bullet \vartriangle l \cdot \phi \cdot \sigma'$	if $(\nu\widetilde{l})(\phi \cdot \bullet\vartriangle l \cdot \sigma') \neq \bot$
(L3)	$\sigma \cdot (\nu\widetilde{l})\sigma'$	$\leq_0$	$\sigma \cdot (\nu\widetilde{l}) \bullet \vartriangle l \cdot \overline{\bullet\vartriangle l} \cdot \sigma'$	if $(\nu\widetilde{l})\sigma' \neq \bot$
(L4)	$\sigma \cdot (\nu l')\phi \cdot \triangleright l' \cdot \sigma'$	$\leq_0$	$\sigma \cdot (\nu l')\phi \cdot \sigma'$	
(L5)	$\sigma \cdot \triangleright l \cdot (\nu\widetilde{l}) \bullet \vartriangle l \cdot \sigma'$	$\leq_0$	$\sigma \cdot (\nu\widetilde{l}) \bullet \vartriangle l \cdot \sigma'$	
(L6)	$\sigma \cdot \triangleright l' \cdot (\nu\widetilde{l})\, I \,@\, l \cdot \sigma'$	$\leq_0$	$\sigma \cdot (\nu\widetilde{l})\, I \,@\, l \cdot \triangleright l' \cdot \sigma'$	if $l' \notin \widetilde{l}$

In laws (L1), (L2), (L3) and (L5), $\bullet\vartriangle l$ stands for either $\triangleright l$ or $l' \triangleleft l$
(hence, $\overline{\bullet\vartriangle l}$ in rule (L3) stands for $\mathbf{nil} \,@\, l$ or $\langle l' \rangle \,@\, l$, respectively)
In rule (L4), ϕ can only be $l' \triangleleft l$ or $\langle l' \rangle \,@\, l$

Table 6: The Ordering Relation on Traces

that permits distinguishing the reception of a free name from the reception of a bound name (this is akin to the asynchronous π-calculus in [9]). In the latter case, the received name l_2 must be fresh for the receiving net and, because of law (RNode), it can be considered as the address of a node; of course, $bn((\nu\widetilde{l})\ l' \triangleleft l\ ) = \widetilde{l}$. Notice that rule (LTS-Rcv) is not needed by the bisimulation we introduced in the previous section to capture barbed congruence. Thus, the new transition system exploits the following labels:

$$\mu \ ::= \ \tau \ \big| \ \phi \qquad\qquad \phi \ ::= \ (\nu\widetilde{l})\, I \,@\, l \ \big| \ \triangleright l \ \big| \ (\nu\widetilde{l})\, l' \triangleleft l$$

where ϕ collects together all the visible labels. Clearly, rules (LTS-Res), (LTS-Par) and (LTS-Struct) from Table 5 must now exploit μ instead of α. Then, we define a complementation function over the labels of the LTS. Formally,

$$\overline{\triangleright l} = \mathbf{nil} \,@\, l \qquad\qquad \overline{\mathbf{nil} \,@\, l} = \triangleright l$$
$$\overline{(\nu\widetilde{l})\, l_2 \triangleleft l_1} = (\nu\widetilde{l})\, \langle l_2 \rangle \,@\, l_1 \qquad\qquad \overline{(\nu\widetilde{l})\, \langle l_2 \rangle \,@\, l_1} = (\nu\widetilde{l})\, l_2 \triangleleft l_1$$

We let σ to range over (possibly empty) sequences of visible actions, i.e.

$$\sigma \ ::= \ \epsilon \ \big| \ \phi \cdot \sigma$$

where ϵ denotes the empty sequence of actions and '$\cdot$' represents concatenation. As usual, $N \overset{\epsilon}{\Longrightarrow}$ denotes $N \Rightarrow$ and $N \overset{\phi\cdot\sigma}{\Longrightarrow}$ denotes $N \overset{\phi}{\Longrightarrow} \overset{\sigma}{\Longrightarrow}$. A naive formulation of trace equivalence such as "$N \overset{\sigma}{\Longrightarrow}$ if and only if $M \overset{\sigma}{\Longrightarrow}$" would be too strong in an asynchronous setting: for example, it would distinguish $N \triangleq l :: \mathbf{in}(!x)@l_1.\mathbf{in}(!y)@l_2$ and $M \triangleq l :: \mathbf{in}(!y)@l_2.\mathbf{in}(!x)@l_1$, which are instead may testing equivalent. Like in [9], a weaker trace equivalence can be defined as follows.

Definition 5.1 (Trace Equivalence) *$\asymp$ is the largest symmetric relation between* cKlaim *nets such that, whenever $N \asymp M$, it holds that $N \overset{\sigma}{\Longrightarrow}$ implies $M \overset{\sigma'}{\Longrightarrow}$, for some $\sigma' \preceq \sigma$.*

The crux is to identify a proper ordering on the traces such that may testing is exactly captured by $\asymp$. The ordering $\preceq$ is obtained as the reflexive and transitive closure of the ordering $\preceq_0$ defined in Table 6. The intuition beyond $\sigma' \preceq \sigma$ is that, if a context can interact with a net that exhibits σ, then the context can interact with any net that exhibits σ'. The ordering $\preceq_0$ relies on the function $(\nu\widetilde{l})\sigma$, that is used in laws (L1), (L2) and (L3) when moving/removing a label of the form $(\nu l')\ l' \triangleleft\ l$. In this case, the information that l' is a fresh received value must be kept in the remaining trace. The formal definition is

$$(\nu\widetilde{l})\sigma \ = \ \begin{cases} \sigma & \text{if } \widetilde{l} \cap fn(\sigma) = \emptyset \\ \sigma_1 \cdot (\nu l')\ l' \triangleleft\ l'' \cdot \sigma_2 & \text{if } \widetilde{l} = \{l'\} \text{ and } \sigma = \sigma_1 \cdot\ l' \triangleleft\ l'' \cdot \sigma_2 \text{ and } l' \notin fn(\sigma_1, l'') \\ \bot & \text{otherwise} \end{cases}$$

To better understand the motivations underlying this definition, consider the following example that justifies the side condition of law (L1) (similar arguments also hold for laws (L2) and (L3)). In the trace $(\nu l')\ l' \triangleleft\ l \cdot \langle l' \rangle\ @\ l''$ performed by the net N, the input action cannot be erased. Indeed, since l' is fresh (see the meaning of label $(\nu l')\ l' \triangleleft\ l$), N cannot get knowledge of l' without performing the input and, consequently, cannot perform the action $\langle l' \rangle\ @\ l''$. On the other hand, if N could receive l' from a communication with another node l''' (thus, it can perform action $l' \triangleleft\ l'''$ after $l' \triangleleft\ l$), then the first input action can be erased and $(\nu l')\ l' \triangleleft\ l''' \cdot \langle l' \rangle\ @\ l'' \ \preceq_0\ (\nu l')\ l' \triangleleft\ l \cdot l' \triangleleft\ l''' \cdot \langle l' \rangle\ @\ l''$.

The intuition beyond the rules in Table 6 now follows. The first three laws have been inspired by [9], while the last three ones are strictly related to the difference between a ‘pure’ name and a name that is used as a node address. Law (L1) states that an input, an output or a migration cannot be directly observed; at most, the *effect* of an output can be observed (by accessing the datum produced by the output). Law (L2) states that the execution of an input/output/migration can be delayed along computations without being noticed by any observer. Law (L3) states that two adjacent ‘complementary’ actions can be deleted (by using a terminology burrowed from CCS [32], we say that ϕ and ϕ' are complementary if they can synchronize to yield a τ – see rules (LTS-Comm) and (LTS-Send)). Law (L4) states that, since bound names can always be used as node addresses, then the reception/transmission of a bound name l' enables outputs/migrations to l'. Law (L5) states that an input from l always enables outputs/migrations to l; indeed, if a datum from l has been retrieved, then l exists and any output/migration to it is enabled. Of course, an output/migration to l always enables other outputs/migrations to l. Similarly, law (L6) states that, if an output/migration to l' is enabled after an action ϕ of the form $(\nu\widetilde{l})\ I\ @\ l$, then the output/migration can be fired before ϕ, since l' was already present. However, this is not possible if l' has been created before ϕ and ϕ extruded it (i.e., if $l' \in \widetilde{l}$).

Remarkably, may testing in the (synchronous/asynchronous) π-calculus [8, 9] cannot distinguish bound names from free ones; thus, a bound name can be replaced with any name in a trace. This is *not* the case here: indeed, bound names can be always considered as addresses of nodes, while free names cannot. This makes a difference for an external observer; thus, a law like

$$\sigma \cdot\ \langle l'' \rangle\ @\ l\ \cdot (\sigma'[^{l''}\!/_{l'}])\ \preceq_0\ \sigma \cdot (\nu l')\langle l' \rangle\ @\ l\ \cdot \sigma'$$

(that, mutatis mutandis, holds for the asynchronous π-calculus [9]) does not hold for cKlaim.

5.1 Soundness w.r.t. May Testing

To prove that trace equivalence exactly captures may testing we rely on the classical definition of the latter equivalence [22], as proposed in Definition 3.4. By using the LTS, $\overset{OK}{\Longrightarrow}$ corresponds to $\xRightarrow{\langle \mathtt{test} \rangle\ @\ \mathtt{test}}$; when it is convenient, we still use OK to denote label $\langle \mathtt{test} \rangle\ @\ \mathtt{test}$.

First, we extend the complementation of a label to traces as expectable:

$$\overline{\sigma} \ = \ \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ \overline{\phi} \cdot \overline{\sigma'} & \text{if } \sigma = \phi \cdot \sigma' \end{cases}$$

Remarkably, $\overline{\overline{\sigma}} = \sigma$. Then, we give a Lemma that describes a sufficient and a necessary condition for the success of an experiment.

Lemma 5.2 *Let N be a net and K be an observer. Then*

1. $N \overset{\sigma}{\Longrightarrow}$ *and* $K \overset{\overline{\sigma}\cdot OK}{\Longrightarrow}$ *imply that* $N \parallel K \overset{OK}{\Longrightarrow}$ *;*
2. $N \parallel K \overset{OK}{\Longrightarrow}$ *implies that there exists a σ such that* $N \overset{\sigma}{\Longrightarrow}$ *and* $K \overset{\overline{\sigma}\cdot OK}{\Longrightarrow}$ *.*

Proof:

1. The proof is by induction on the lenght of σ. The base step is trivial. For the inductive step, we have that $\sigma = \phi\cdot\sigma'$ and we consider the possibilities for ϕ. All the cases are trivial, except for the following two:

   - $\phi = (\nu l')\ l' \triangleleft l$. Now, we have that $N \Rightarrow N' \xrightarrow{(\nu l')\ l' \triangleleft l} N'' \overset{\sigma'}{\Longrightarrow}$, for $l' \notin fn(N')$; moreover, $K \Rightarrow K' \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} K'' \overset{\overline{\sigma'}\cdot OK}{\Longrightarrow}$. By induction, $N'' \parallel K'' \overset{OK}{\Longrightarrow}$. Now, by Proposition 4.3.3, $K' \equiv (\nu l')(K''' \parallel l :: \langle l'\rangle)$ and $K'' \equiv K''' \parallel l ::$ **nil**; thus, $N \parallel K \Rightarrow (\nu l')(N' \parallel K''' \parallel l :: \langle l'\rangle) \xrightarrow{\tau} (\nu l')(N'' \parallel K'') \overset{OK}{\Longrightarrow}$ (indeed, since K is an observer, it can only emit `test` at `test`; thus, $l' \neq$ `test`).
   - $\phi = (\nu l')\ \langle l'\rangle\ @\ l$. This case is similar.

2. By definition, it must be that $N \parallel K(\xrightarrow{\tau})^n H \xrightarrow{OK}$; the proof is by induction on n. The base step is simple: $\sigma = \epsilon$. For the inductive step, we have two subcases:

   - $N \parallel K \xrightarrow{\tau} N' \parallel K'(\xrightarrow{\tau})^{n-1}H$. By induction, $N' \overset{\sigma'}{\Longrightarrow}$ and $K' \overset{\overline{\sigma'}\cdot OK}{\Longrightarrow}$, for some σ'. There are six possibilities for the first τ-step:
     (a) $N \xrightarrow{\tau} N'$ and $K' \equiv K$: in this case, we can pick up $\sigma = \sigma'$.
     (b) $N' \equiv N$ and $K \xrightarrow{\tau} K'$: like before.
     (c) $N \xrightarrow{\triangleright l} N'$ and $K \xrightarrow{\textbf{nil}\ @\ l} K'$: we can pick up $\sigma = \triangleright l \cdot \sigma'$.
     (d) $N \xrightarrow{\textbf{nil}\ @\ l} N'$ and $K \xrightarrow{\triangleright l} K'$: we can pick up $\sigma =$ **nil** $@\ l \cdot \sigma'$.
     (e) $N \xrightarrow{l' \triangleleft l} N'$ and $K \xrightarrow{\langle l'\rangle\ @\ l} K'$: we can pick up $\sigma = l' \triangleleft l \cdot \sigma'$.
     (f) $N \xrightarrow{\langle l'\rangle\ @\ l} N'$ and $K \xrightarrow{l' \triangleleft l} K'$: we can pick up $\sigma = \langle l'\rangle\ @\ l \cdot \sigma'$.
   - $N \parallel K \xrightarrow{\tau} (\nu l')(N' \parallel K')(\xrightarrow{\tau})^{n-1}H$. Since $H \equiv (\nu l')H'$ for some H' (indeed, τ-steps cannot remove restrictions), it must be $l' \neq$ `test`. Thus, $N' \parallel K'(\xrightarrow{\tau})^{n-1}H' \xrightarrow{OK}$ and, by induction, $N' \overset{\sigma'}{\Longrightarrow}$ and $K' \overset{\overline{\sigma'}\cdot OK}{\Longrightarrow}$, for some σ'. There are only two possibilities for the first τ-step:
     (a) $N \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} N'$ and $K \xrightarrow{l' \triangleleft l} K'$. By definition of the LTS, we have to extend the scope of l' before passing it by using rule (Ext). Thus, $l' \notin fn(K)$ and, by rule (LTS-Rcv), we have that $K \xrightarrow{(\nu l')\ l' \triangleleft l} K'$. Thus, we can pick up $\sigma = (\nu l')\langle l'\rangle\ @\ l \cdot \sigma'$.
     (b) $N \xrightarrow{l' \triangleleft l} N'$ and $K \xrightarrow{(\nu l')\ \langle l'\rangle\ @\ l} K'$: this case is similar. ∎

The next Lemma states that the laws in Table 6 are 'sound', in the sense that, if an observer can observe a trace σ (i.e., that can provide $\overline{\sigma}$), then it can also observe any trace $\sigma' \preceq \sigma$.

Lemma 5.3 *Let $\sigma' \leq \sigma$ and $N \overset{\overline{\sigma}}{\Longrightarrow}$; then, $N \overset{\overline{\sigma'}}{\Longrightarrow}$.*

Proof: By definition, $\sigma'(\leq_0)^n\sigma$; we proceed by induction on n. The base step is trivial, by reflexivity. For the inductive step, we let $\sigma'(\leq_0)^{n-1}\sigma'' \leq_0 \sigma$; it suffices to prove that $N \overset{\overline{\sigma}}{\Longrightarrow}$ implies that $N \overset{\overline{\sigma''}}{\Longrightarrow}$. Indeed, by induction, the latter judgement implies that $N \overset{\overline{\sigma'}}{\Longrightarrow}$, as required. We reason by case analysis on the law in Table 6 used to infer $\sigma'' \leq_0 \sigma$. Notice that all the laws hide a double formulation that is made explicit in this proof.

(L1).a: $\sigma \triangleq \sigma_1 \cdot (\nu\widetilde{l})\, l' \triangleleft l \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu\widetilde{l})\sigma_2$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu\widetilde{l})\,\langle l'\rangle\,@\,l} N'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N' \equiv (\nu\widetilde{l})(N''' \parallel l :: \langle l'\rangle)$ and $N'' \equiv N''' \parallel l :: \mathbf{nil}$. Now, if $\widetilde{l} = \emptyset$ or $\widetilde{l} \cap fn(\sigma_2) = \emptyset$, then it must be that $N' \overset{\overline{\sigma_2}}{\Longrightarrow}$ and, hence, $N \overset{\overline{\sigma''}}{\Longrightarrow}$. Otherwise, it must be $\widetilde{l} = \{l'\}$ and $\sigma_2 \triangleq \sigma_3 \cdot l' \triangleleft l'' \cdot \sigma_4$ for $l' \notin fn(\sigma_3, l'')$; thus, $N'' \overset{\overline{\sigma_3}}{\Longrightarrow} N''_1 \xrightarrow{\langle l'\rangle\,@\,l''} N''_2 \overset{\overline{\sigma_4}}{\Longrightarrow}$. Now, $N' \overset{\overline{\sigma_3}}{\Longrightarrow} (\nu l')(N''_1 \parallel l :: \langle l'\rangle \parallel l'' :: \langle l'\rangle)$ and hence $N \xRightarrow{\overline{\sigma_1}\cdot\overline{\sigma_3}\cdot(\nu l')\,\langle l'\rangle\,@\,l''\,\cdot\overline{\sigma_4}}$, i.e. $N \overset{\overline{\sigma''}}{\Longrightarrow}$.

(L1).b: $\sigma \triangleq \sigma_1 \cdot \triangleright l \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \sigma_2$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{\mathbf{nil}\,@\,l} N'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N' \equiv N''$ and hence $N \overset{\overline{\sigma_1}\cdot\overline{\sigma_2}}{\Longrightarrow}$, as required.

(L2).a: $\sigma \triangleq \sigma_1 \cdot (\nu\widetilde{l})\, l' \triangleleft l \cdot \phi \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu\widetilde{l})(\phi \cdot l' \triangleleft l \cdot \sigma_2)$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu\widetilde{l})\,\langle l'\rangle\,@\,l} N'' \overset{\overline{\phi}}{\Longrightarrow} N''' \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N' \equiv (\nu\widetilde{l})(\hat{N} \parallel l :: \langle l'\rangle)$ and $N'' \equiv \hat{N} \parallel l :: \mathbf{nil}$. Now, if $\widetilde{l} = \emptyset$ or $\widetilde{l} \cap fn(\phi) = \emptyset$, then it must be that $N' \xRightarrow{\overline{\phi}\cdot(\nu\widetilde{l})\,\langle l'\rangle\,@\,l\,\cdot\overline{\sigma_2}}$ and, hence, $N \overset{\overline{\sigma''}}{\Longrightarrow}$. Otherwise, it must be $\phi = l' \triangleleft l''$ for $l' \neq l''$; thus, $N'' \xRightarrow{\langle l'\rangle\,@\,l''} N'''$. Now, $N' \xRightarrow{(\nu l')\,\langle l'\rangle\,@\,l''} N''' \parallel l :: \langle l'\rangle \xrightarrow{\langle l'\rangle\,@\,l} N''' \parallel l :: \mathbf{nil} \overset{\overline{\sigma_2}}{\Longrightarrow}$, and hence $N \overset{\overline{\sigma''}}{\Longrightarrow}$.

(L2).b: $\sigma \triangleq \sigma_1 \cdot \triangleright l \cdot \phi \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \phi \cdot \triangleright l \cdot \sigma_2$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{\mathbf{nil}\,@\,l} N'' \overset{\overline{\phi}}{\Longrightarrow} N''' \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N' \equiv N'' \equiv N'' \parallel l :: \mathbf{nil}$. This implies that $N''' \equiv N''' \parallel l :: \mathbf{nil}$ (since nodes cannot disappear along reductions) and $N \xRightarrow{\overline{\sigma_1}\cdot\overline{\phi}\cdot\ \mathbf{nil}\,@\,l\ \cdot\cdot\overline{\sigma_2}}$, as required.

(L3).a: $\sigma \triangleq \sigma_1 \cdot (\nu\widetilde{l})\, l' \triangleleft l \cdot \langle l'\rangle\,@\,l \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu\widetilde{l})\sigma_2$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu\widetilde{l})\,\langle l'\rangle\,@\,l} N'_1 \Rightarrow N'_2 \xrightarrow{l' \triangleleft l} N'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N' \equiv (\nu\widetilde{l})(\hat{N} \parallel l :: \langle l'\rangle)$ and $N'_1 \equiv \hat{N} \parallel l :: \mathbf{nil}$. Thus, $N' \Rightarrow (\nu\widetilde{l})(N'_2 \parallel l :: \langle l'\rangle) \xrightarrow{\tau} (\nu\widetilde{l})(N'' \parallel l :: \mathbf{nil}) \triangleq N_3$. Now, if $\widetilde{l} = \emptyset$ or $\widetilde{l} \cap fn(\sigma_2) = \emptyset$, then $N_3 \overset{\overline{\sigma_2}}{\Longrightarrow}$ and, hence, $N \overset{\overline{\sigma''}}{\Longrightarrow}$. Otherwise, we reason like in case (L1).a to obtain that $N_3 \overset{(\nu\widetilde{l})\overline{\sigma_2}}{\Longrightarrow}$ and, again, $N \overset{\overline{\sigma''}}{\Longrightarrow}$.

(L3).b: $\sigma \triangleq \sigma_1 \cdot \triangleright l \cdot \mathbf{nil}\,@\,l \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \sigma_2$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{\mathbf{nil}\,@\,l} N'' \overset{\triangleright l}{\Longrightarrow} N''' \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N' \equiv N'' \parallel l :: \mathbf{nil}$. This implies that $N' \Rightarrow N'''$ and we can easily conclude.

(L4).a: $\sigma \triangleq \sigma_1 \cdot (\nu l')\, l' \triangleleft l \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu l')\, l' \triangleleft l \cdot \triangleright l' \cdot \sigma_2$. Then, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu l')\,\langle l'\rangle\,@\,l} N'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; thus, by rule (RNode), $N' \equiv (\nu l')(\hat{N} \parallel l :: \langle l'\rangle \parallel l' :: \mathbf{nil})$ and $N'' \equiv \hat{N} \parallel l :: \mathbf{nil}$. Hence, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu l')\,\langle l'\rangle\,@\,l} \hat{N} \parallel l :: \mathbf{nil} \parallel l' :: \mathbf{nil} \xrightarrow{\mathbf{nil}\,@\,l'} \overset{\overline{\sigma_2}}{\Longrightarrow}$, as needed.

(L4).b: $\sigma \triangleq \sigma_1 \cdot (\nu l')\ \langle l' \rangle\ @\ l\ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu l')\ \langle l' \rangle\ @\ l\ \cdot\ \triangleright l'\ \cdot \sigma_2$. Then, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu l')\ l' \triangleleft l} N'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; by rule (LTS-Rcv), it holds that $N'' \equiv N'' \parallel l' :: \mathbf{nil}$. Thus, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu l')\ l' \triangleleft l} N'' \parallel l' :: \mathbf{nil} \xrightarrow{\mathbf{nil}\ @\ l'} \overset{\overline{\sigma_2}}{\Longrightarrow}$, as needed.

(L5).a: $\sigma \triangleq \sigma_1 \cdot (\nu\widetilde{l})\ l' \triangleleft l\ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot\ \triangleright l\ \cdot (\nu\widetilde{l})\ l' \triangleleft l\ \cdot \sigma_2$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{(\nu\widetilde{l})\ \langle l' \rangle\ @\ l} N'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N' \equiv (\nu\widetilde{l})(N''' \parallel l :: \langle l' \rangle)$ and $N'' \equiv N''' \parallel l :: \mathbf{nil}$. Thus, easily, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N' \xrightarrow{\mathbf{nil}\ @\ l} \xrightarrow{(\nu\widetilde{l})\ \langle l' \rangle\ @\ l} N'' \overset{\overline{\sigma_2}}{\Longrightarrow}$, as required.

(L5).b: $\sigma \triangleq \sigma_1 \cdot\ \triangleright l\ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot\ \triangleright l\ \cdot\ \triangleright l\ \cdot \sigma_2$. Similar to case (L5).a.

(L6).a: $\sigma \triangleq \sigma_1 \cdot (\nu\widetilde{l})\ \langle l'' \rangle\ @\ l\ \cdot\ \triangleright l'\ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot\ \triangleright l'\ \cdot (\nu\widetilde{l})\ \langle l'' \rangle\ @\ l\ \cdot \sigma_2$. By definition, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N_1 \xrightarrow{(\nu\widetilde{l})\ l'' \triangleleft l} N_2 \Rightarrow N_3 \xrightarrow{\mathbf{nil}\ @\ l'} N_4 \overset{\overline{\sigma_2}}{\Longrightarrow}$; moreover, $N_3 \equiv N_3 \parallel l' :: \mathbf{nil} \equiv N_4$. Since $l' \notin \widetilde{l}$, we have that l' must be a node also in N_2 and in N_1, i.e., $N_2 \equiv N_2 \parallel l' :: \mathbf{nil}$ and $N_1 \equiv N_1 \parallel l' :: \mathbf{nil}$. Hence, $N \overset{\overline{\sigma_1}}{\Longrightarrow} N_1 \xrightarrow{\mathbf{nil}\ @\ l'} \xrightarrow{(\nu\widetilde{l})\ \langle l' \rangle\ @\ l} N_2 \Rightarrow N_4 \overset{\overline{\sigma_2}}{\Longrightarrow}$, as required.

(L6).b: $\sigma \triangleq \sigma_1 \cdot\ \mathbf{nil}\ @\ l\ \cdot\ \triangleright l'\ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot\ \triangleright l'\ \cdot\ \mathbf{nil}\ @\ l\ \cdot \sigma_2$. Similar to case (L6).a. ∎

Now, the main theorem follows.

Theorem 5.4 (Soundness of $\asymp$ w.r.t. $\simeq$) *If $N \asymp M$ then $N \simeq M$.*

Proof: Let K be an oserver such that $N \parallel K \overset{OK}{\Longrightarrow}$. By Lemma 5.2.2, there exists σ such that $N \overset{\sigma}{\Longrightarrow}$ and $K \overset{\overline{\sigma}\cdot OK}{\Longrightarrow}$. By Definition 5.1, there exists $\sigma' \preceq \sigma$ such that $M \overset{\sigma'}{\Longrightarrow}$. By suffix closure of $\preceq$ (that can be easily proved), we have that $\sigma' \cdot \texttt{test} \triangleleft \texttt{test} \preceq \sigma \cdot \texttt{test} \triangleleft \texttt{test}$. By Lemma 5.3, $K \overset{\overline{\sigma'}\cdot OK}{\Longrightarrow}$. By Lemma 5.2.1, $M \parallel K \overset{OK}{\Longrightarrow}$, as required by Definition 3.4. Thus, $N \simeq' M$ that, by Proposition 3.5, implies that $N \simeq M$. ∎

5.2 Completeness w.r.t. May Testing

Now, we define the *canonical observer* for a tace σ, written $\amalg(\sigma)$, as

$$\amalg(\sigma) \;=\; C[\texttt{test} :: P]$$

where the actual observer process P and context $C[\cdot]$ enabling the observation are returned by $O_{\emptyset}(\sigma) = \,<P\,;\,C[\cdot]>$, which is inductively defined as follows

$$
\begin{array}{rcl}
O_L(\epsilon) & = & <\mathbf{out}(\mathtt{test})@\mathtt{test}.\mathbf{nil}\,;\,[\cdot]> \\[1ex]
O_L(\,\mathbf{nil}\ @\ l\ \cdot \sigma) & = & <\mathbf{eval}(\mathbf{nil})@l.P\,;\,C[\cdot]> \\
 & & \text{where } O_L(\sigma) = \,<P\,;\,C[\cdot]> \\[1ex]
O_L(\,\langle l'\rangle\ @\ l\ \cdot \sigma) & = & <\mathbf{in}(l')@l.P\,;\,C[\cdot]> \\
 & & \text{where } O_L(\sigma) = \,<P\,;\,C[\cdot]> \\[1ex]
O_L((\nu l')\,\langle l'\rangle\ @\ l\ \cdot \sigma) & = & <\mathbf{in}(!x)@l.(P[{}^{x}\!/\!{}_{l'}])\,;\,C[\cdot]> \\
 & & \text{where } O_{L\cup\{l'\}}(\sigma) = \,<P\,;\,C[\cdot]> \\[1ex]
O_L(\,\triangleright\, l\ \cdot \sigma) & = & \begin{cases} <P\,;\,C[\cdot]\ \|\ l :: \mathbf{nil}> & \text{if } l \notin L \\[1ex] <P\,;\,C[\cdot]> & \text{otherwise} \end{cases} \\
 & & \text{where } O_L(\sigma) = \,<P\,;\,C[\cdot]> \\[1ex]
O_L((\nu\widetilde{l})\,l'\ \triangleleft\ l\ \cdot \sigma) & = & \begin{cases} <\mathbf{new}(\widetilde{l}).\mathbf{out}(l')@l.P\,;\,C[\cdot]\ \|\ l :: \mathbf{nil}> & \text{if } l \notin L \\[1ex] <\mathbf{new}(\widetilde{l}).\mathbf{out}(l')@l.P\,;\,C[\cdot]> & \text{otherwise} \end{cases} \\
 & & \text{where } O_L(\sigma) = \,<P\,;\,C[\cdot]>
\end{array}
$$

L is the (finite) set of names extruded by the trace, i.e. those names created by the net that emitted σ and offered as a datum in a visible location. We used the convention that $\mathbf{new}(\widetilde{l}).\mathbf{out}(l')@l$ stands for $\mathbf{out}(l')@l$ whenever $\widetilde{l} = \emptyset$. The context has only to provide localities where

- the observed net can place data/code (when σ is of the form $\triangleright\, l\ \cdot \sigma'$)
- the observer process can place data that the observed net needs (when σ is of the form $(\nu\widetilde{l})\,l'\ \triangleleft\ l\ \cdot \sigma'$).

However, the context should not provide a locality l' whenever $l' \in L$. In this case, the observed net already provides l'; indeed, if $l' \in L$, then l' has been extruded by an action $(\nu l')\,\langle l'\rangle\ @\ l$ in σ.

The key property of the canonical observer for σ is that it always reports success when run in parallel with a net that offers σ, as stated by the following Proposition.

Proposition 5.5 $\amalg(\sigma) \xRightarrow{\overline{\sigma}\cdot OK}$.

Proof: The proof is by induction on $|\sigma|$ and easily follows by definition of canonical observers. ∎

Now, we distinguish the label $\triangleright\, l$ generated by rule (LTS-Out) from the same label generated by rule (LTS-Eval). We shall write $\boxed{\triangleright}\, l$ the former and $\triangleright\, l$ the latter. This is needed for technical reasons (see the case (**iv**) in the proof of Lemma 5.7); the two labels are exactly the same. We start by adapting Lemma 5.2.2 in order to exclude labels of the form $\boxed{\triangleright}\, l$.

Lemma 5.6 *If $N \parallel \amalg(\sigma) \xRightarrow{OK}$, then there exists a σ' such that $N \xRightarrow{\sigma'}$, $\amalg(\sigma) \xRightarrow{\overline{\sigma'}\cdot OK}$ and σ' does not contain labels of the form $\boxed{\triangleright}\, l$.*

Proof: By Lemma 5.2.2, we know that there exists a trace σ'' such that $N \xRightarrow{\sigma''}$ and $\amalg(\sigma) \xRightarrow{\overline{\sigma''}\cdot OK}$. The proof now proceeds by induction on the number of labels of the form $\boxed{\triangleright}\, l$ (for a generic l) in $\overline{\sigma''}$. The base step is trivial. For the inductive step, we have that $\overline{\sigma''} = \overline{\sigma_1} \cdot \boxed{\triangleright}\, l \cdot \overline{\sigma_2}$ such that $\overline{\sigma_1}$ does not contain labels of the form $\boxed{\triangleright}\, \_$. We consider two cases:

- There are no *intruded* names[3] in $\overline{\sigma_1}$. Thus, σ_1 does not contain labels of the form $(\nu l')\ \langle l' \rangle\ @\ l''$. Let $\text{Ш}(\sigma) \triangleq C[\texttt{test} :: P] \overset{\overline{\sigma_1}}{\Longrightarrow} C'[\texttt{test} :: \mathbf{out}(l')@l.P] \xrightarrow{\boxed{\triangleright}\ l} C'[\texttt{test} :: P] \parallel l :: \langle l' \rangle \overset{\overline{\sigma_2}}{\Longrightarrow}$. By definition of canonical observers, it must be that $C'[\cdot] \equiv C'[\cdot] \parallel l :: \mathbf{nil}$; indeed, for every action **out** at l in a canonical observer, a node with address l is always provided, except when l is a name intruded by the observer (i.e., extruded by the trace), that is not the case here. Thus, $\text{Ш}(\sigma) \overset{\overline{\sigma_1}\cdot\overline{\sigma_2}}{\Longrightarrow}$. On the other side, $N \overset{\sigma_1}{\Longrightarrow} N' \xrightarrow{\mathbf{nil}\ @\ l} N'' \overset{\sigma_2}{\Longrightarrow}$, where $N' \equiv N''$; thus, $N \overset{\sigma_1\cdot\sigma_2}{\Longrightarrow}$. The thesis holds by induction on $\overline{\sigma_1} \cdot \overline{\sigma_2}$ that has one label of the form $\boxed{\triangleright}\ \_$ less than $\overline{\sigma''}$.

- There are intruded names in $\overline{\sigma_1}$ and these are $\{l_1, \ldots, l_k\}$. If $l \notin \{l_1, \ldots, l_k\}$, then the proof is like in the case above; otherwise, let l be l_i. Since l_i has been intruded, it must be that $\text{Ш}(\sigma) \overset{\overline{\sigma_3}}{\Longrightarrow} K \xrightarrow{(\nu l_i)\ l_i\ \triangleleft\ l''} K' \parallel l_i :: \mathbf{nil} \overset{\overline{\sigma_4}}{\Longrightarrow} C[\texttt{test} :: \mathbf{out}(l')@l_i.P]$, where $\sigma_1 = \sigma_3 \cdot (\nu l_i)\ \langle l_i \rangle\ @\ l'' \cdot \sigma_4$ and $C[\texttt{test} :: P] \parallel l_i :: \langle l' \rangle \overset{\overline{\sigma_2}}{\Longrightarrow}$. Now, $N \overset{\sigma_3}{\Longrightarrow} N_1 \xrightarrow{(\nu l_i)\ \langle l_i \rangle\ @\ l''} N_2 \overset{\sigma_4}{\Longrightarrow} N_3 \xrightarrow{\mathbf{nil}\ @\ l_i} N_4 \overset{\sigma_2}{\Longrightarrow}$, for some $N_2 \equiv N_2 \parallel l_i :: \mathbf{nil}$ (this is always possible because in N_1 name l_i is restricted and can be used as address of a node, by using law (RNode)). Moreover, $N_3 \equiv N_4$; thus, since the node with address l_i in $K' \parallel l_i :: \mathbf{nil}$ cannot disappear during computations, it holds that $\text{Ш}(\sigma) \overset{\overline{\sigma_3}\cdot(\nu l_i)\ l_i\ \triangleleft\ l''\ \cdot\overline{\sigma_4}\cdot\overline{\sigma_2}}{\Longrightarrow}$, i.e. $\text{Ш}(\sigma) \overset{\overline{\sigma_1}\cdot\overline{\sigma_2}}{\Longrightarrow}$, and correspondingly $N \overset{\sigma_1\sigma_2}{\Longrightarrow}$. Like before, we can apply induction to $\overline{\sigma_1} \cdot \overline{\sigma_2}$ and conclude. ∎

The main Lemma to prove completeness of trace equivalence w.r.t. may testing is the following one, stating that $\text{Ш}(\sigma)$ can report success only upon execution of a trace σ' such that $\overline{\sigma'} \preceq \sigma$.

Lemma 5.7 *Let* $\text{Ш}(\sigma) \overset{\sigma'\cdot OK}{\Longrightarrow}$ *, where* $\texttt{test} \notin n(\sigma')$ *and* σ' *does not contain labels of the form* $\boxed{\triangleright}\ l$*. Then,* $\overline{\sigma'} \preceq \sigma$*.*

Proof: The proof is by induction on $|\sigma|$. The base step is trivial. For the inductive step, let σ be $\phi \cdot \sigma''$; let us reason on the possibilities for ϕ.

(i) $\phi \triangleq \mathbf{nil}\ @\ l$ **.** By costruction, $\text{Ш}(\sigma) \triangleq C[\texttt{test} :: \mathbf{eval}(\mathbf{nil})@l.P]$, where $< P\ ;\ C[\cdot] > = O_\emptyset(\sigma'')$. The trace $\text{Ш}(\sigma) \overset{\sigma'\cdot OK}{\Longrightarrow}$ can be produced only in two ways:

1. $\sigma' \triangleq \sigma_1 \cdot \triangleright l \cdot \sigma_2$, where $C[\mathbf{0}] \overset{\sigma_1}{\Longrightarrow} C'[\mathbf{0}]$ and $C'[\texttt{test} :: P] \overset{\sigma_2\cdot OK}{\Longrightarrow}$. Thus, $\text{Ш}(\sigma'') \triangleq C[\texttt{test} :: P] \overset{\sigma_1\cdot\sigma_2\cdot OK}{\Longrightarrow}$; by induction, this implies that $\overline{\sigma_1 \cdot \sigma_2} \preceq \sigma''$. Now, $\sigma \triangleq \phi \cdot \sigma'' \succeq \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2} \succeq \overline{\sigma_1} \cdot \phi \cdot \overline{\sigma_2} \triangleq \overline{\sigma'}$, where the first inequality holds by prefix closure of $\succeq$ (i.e., the inverse of $\preceq$) while the second inequality holds by repeated applications of law (L6). Indeed, since $C[\cdot]$ is just a parallel of nodes with no components, σ_1 only contains labels of the form $\mathbf{nil}\ @\ \_$; thus, $\overline{\sigma_1}$ only contains labels of the form $\triangleright\ \_$.

2. $\sigma' \triangleq \sigma_1 \cdot \sigma_2$, where $C[\mathbf{0}] \overset{\sigma_1\cdot\ \mathbf{nil}\ @\ l}{\Longrightarrow} C'[\mathbf{0}]$ and $C'[\texttt{test} :: P] \overset{\sigma_2\cdot OK}{\Longrightarrow}$. Thus, $\text{Ш}(\sigma'') \triangleq C[\texttt{test} :: P] \overset{\sigma_1\cdot\ \mathbf{nil}\ @\ l\ \cdot\sigma_2\cdot OK}{\Longrightarrow}$; by induction, this implies that $\overline{\sigma_1} \cdot \triangleright l \cdot \overline{\sigma_2} \preceq \sigma''$. Now, by prefix closure, by repeated applications of law (L6) (like in the previous case) and by law (L3), we have that $\sigma \triangleq \phi \cdot \sigma'' \succeq \phi \cdot \overline{\sigma_1} \cdot \triangleright l \cdot \overline{\sigma_2} \succeq \overline{\sigma_1} \cdot \triangleright l \cdot \phi \cdot \overline{\sigma_2} \succeq \overline{\sigma_1} \cdot \overline{\sigma_2} \triangleq \overline{\sigma'}$, as required.

[3]By symmetry of denomination w.r.t. extruded names, we call *intruded* a name received via rule (LTS-Rcv), i.e. name l' is intruded in σ if $\sigma = \sigma' \cdot (\nu l')\ l'\ \triangleleft\ l\ \cdot \sigma''$.

(ii) $\phi \triangleq\ \triangleright l$ **.** By costruction, $\text{Ш}(\sigma) \triangleq C[\texttt{test} :: P] \parallel l :: \textbf{nil}$, where $< P\,;\, C[\cdot] > \ = O_{\emptyset}(\sigma'')$. Now, we have that $\sigma' \triangleq \sigma_1 \cdot \sigma_2$, where $C[\texttt{test} :: P] \overset{\sigma_1}{\Longrightarrow} C'[\texttt{test} :: P']$ and $C'[\texttt{test} :: P'] \parallel l :: \textbf{nil} \overset{\sigma_2 \cdot OK}{\Longrightarrow}$. Now, $\text{Ш}(\sigma'') \triangleq C[\texttt{test} :: P] \overset{\sigma_1 \cdot \sigma'_2 \cdot OK}{\Longrightarrow}$, where σ'_2 is the trace obtained from σ_2 by removing all the labels $\textbf{nil} @ l$ from it and by possibly adding a label of the form $\triangleright l$. Indeed, since it is not necessarily the case that $C[\cdot] \equiv \ldots \parallel l :: \textbf{nil}$, it can be that some labels $\textbf{nil} @ l$ cannot be generated by $C'[\texttt{test} :: P']$; similarly, it could be necessary to add a label $\triangleright l$ if, in the production of σ_2, $C'[\texttt{test} :: P']$ needs l to place some data/process. Hence, by induction, we have that $\overline{\sigma_1} \cdot \overline{\sigma'_2} \preceq \sigma''$. We now have the desired $\sigma \succeq \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma'_2} \succeq \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2} \succeq \overline{\sigma_1} \cdot \overline{\sigma_2}$. Notice that the second inequality has been obtained by repeated applications of laws (L5) and (L2) (as many times as the number of labels $\textbf{nil} @ l$ removed from σ_2 to obtain σ'_2) and by possibly applying laws (L5), (L2) and (L3) (if a label of the form $\triangleright l$ has been introduced in σ'_2). The last inequality relies on law (L1).

(iii) $\phi \triangleq (\nu\widetilde{l})\,\langle l' \rangle @ l$ **.** By costruction, we have two subcases:

1. $\widetilde{l} = \emptyset$. Then, $\text{Ш}(\sigma) \triangleq C[\texttt{test} :: \textbf{in}(l')@l.P]$, where $< P\,;\, C[\cdot] > \ = O_{\emptyset}(\sigma'')$. Now, we have that $\sigma' \triangleq \sigma_1 \cdot\ l' \triangleleft l\ \cdot \sigma_2$, where $C[\texttt{test} :: \textbf{in}(l')@l.P] \overset{\sigma_1}{\Longrightarrow} C'[\texttt{test} :: \textbf{in}(l')@l.P] \xrightarrow{l' \triangleleft l} C'[\texttt{test} :: P] \overset{\sigma_2 \cdot OK}{\Longrightarrow}$. Thus, $\text{Ш}(\sigma'') \overset{\sigma_1 \cdot \sigma_2 \cdot OK}{\Longrightarrow}$ and, by induction, $\overline{\sigma_1} \cdot \overline{\sigma_2} \preceq \sigma''$. Again, by using prefix closure and law (L6) (that can be used since $\overline{\sigma_1}$ is only made up by labels of the form $\triangleright \_$), $\sigma \succeq \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2} \succeq \overline{\sigma_1} \cdot \phi \cdot \overline{\sigma_2} \triangleq \overline{\sigma'}$.
2. $\widetilde{l} = \{l'\}$. Then, $\text{Ш}(\sigma) \triangleq C[\texttt{test} :: \textbf{in}(!x)@l.(P[^{x}/_{l'}])]$, where $< P\,;\, C[\cdot] > \ = O_{\{l'\}}(\sigma'')$. Now, we have that $\sigma' \triangleq \sigma_1 \cdot (\nu l')\ l' \triangleleft l\ \cdot \sigma_2$, where $C[\texttt{test} :: \textbf{in}(!x)@l.(P[^{x}/_{l'}])] \overset{\sigma_1}{\Longrightarrow} C'[\texttt{test} :: \textbf{in}(!x)@l.(P[^{x}/_{l'}])] \xrightarrow{(\nu l')\ l' \triangleleft l} C'[\texttt{test} :: P] \parallel l' :: \textbf{nil} \overset{\sigma_2 \cdot OK}{\Longrightarrow}$ (indeed, by construction, $l' \notin fn(C'[\texttt{test} :: \textbf{in}(!x)@l.(P[^{x}/_{l'}])]))$. By an easy inspection of the definition of canonical observers, it holds that $\text{Ш}(\sigma'')$ is structurally equivalent to either $C[\texttt{test} :: P] \parallel l' :: \textbf{nil}$ or $C[\texttt{test} :: P]$ (according to whether σ'' contains labels of the form $\triangleright l'$ and $\_ \triangleleft l'$ or not). In the first case, $\text{Ш}(\sigma'') \overset{\sigma_1 \cdot \sigma_2 \cdot OK}{\Longrightarrow}$ and the thesis follows by induction, prefix closure and applications of law (L6). In the second case, we proceed like in case (**ii**) above, i.e. $\text{Ш}(\sigma'') \overset{\sigma_1 \cdot \sigma'_2 \cdot OK}{\Longrightarrow}$ where σ'_2 is obtained from σ_2 by removing actions $\textbf{nil} @ l'$ and by possibly adding an action $\triangleright l'$. The proof is then similar, but uses (L6) to place ϕ at its right place.

(iv) $\phi \triangleq (\nu\widetilde{l})\ l' \triangleleft l$ **.** By costruction, $\text{Ш}(\sigma) \triangleq C[\texttt{test} :: \textbf{new}(\widetilde{l}).\textbf{out}(l')@l.P] \parallel l :: \textbf{nil}$, where $< P\,;\, C[\cdot] > \ = O_{\emptyset}(\sigma'')$. This case is the most tedious: $\text{Ш}(\sigma)$ has a lot of possible evolutions and, thus, σ' can be of several forms. However, by hypothesis σ' does not contain labels of the form $\boxed{\triangleright}\ \_$; in particular, it does not contain $\boxed{\triangleright}\ l$. Hence, the action $\textbf{out}(l')@l$ does not generate any visible action and forces $\text{Ш}(\sigma)$ to reduce to $(\nu\widetilde{l})(C[\texttt{test} :: P] \parallel l :: \langle l' \rangle)$ in order to report success. We consider only the case in which $\widetilde{l} = \{l'\}$, that is the most complicated. We have to keep into account whether and how $C[\texttt{test} :: P]$ and $l :: \langle l' \rangle$ interact, and whether and how l' is extruded. We have 12 possibilities in total.

1. *$l :: \langle l' \rangle$ is not involved in the generation of σ'*

   (a) *l' is not extruded by $C[\texttt{test} :: P]$:* this case is the simplest, since $\text{Ш}(\sigma'') \overset{\sigma' \cdot OK}{\Longrightarrow}$. By an easy induction and by using (L1) we can conclude; indeed, notice that $l' \notin fn(\sigma')$ and thus $(\nu l')\sigma' = \sigma'$.

(b) *l' is extruded by C*[`test` :: *P*]*:* in this case, we have that $\sigma' \triangleq \sigma_1 \cdot (\nu l')\,\langle l'\rangle\ @\ l''\ \cdot \sigma_2$ and $\amalg(\sigma'') \xRightarrow{\sigma_1 \cdot\ \langle l'\rangle\ @\ l''\ \cdot \sigma_2 \cdot OK}$. By using induction and prefix closure, we have that $\sigma \geq \phi \cdot \overline{\sigma_1} \cdot\ l' \triangleleft l''\ \cdot \overline{\sigma_2} \geq \overline{\sigma_1} \cdot (\nu l')\, l' \triangleleft l''\ \cdot \overline{\sigma_2} \triangleq \overline{\sigma'}$, where the second inequality relies on law (L1).

2. *the first contribution of l :: $\langle l'\rangle$ in σ' is with label* **nil** $@\ l$

(a) *the datum $\langle l'\rangle$ is not used*

i. *l' is not extruded:* in this case $\sigma' \triangleq \sigma_1 \cdot$ **nil** $@\ l \cdot \sigma_2$ and $\amalg(\sigma'') \xRightarrow{\sigma_1 \cdot \sigma_2' \cdot OK}$, where σ_2' is obtained from σ_2 like in case (**ii**) above. By induction and prefix closure, $\sigma \geq \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2'} \geq\ \triangleright l \cdot \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2'} \geq\ \triangleright l \cdot \overline{\sigma_1} \cdot \overline{\sigma_2'} \geq \overline{\sigma_1} \cdot \triangleright l \cdot \overline{\sigma_2'} \geq \overline{\sigma_1} \cdot \triangleright l \cdot \overline{\sigma_2} \triangleq \overline{\sigma'}$, where the second inequality holds by law (L5), the third inequality holds by law (L1) and because l' does not appear in what follows, the fourth inequality holds by law (L2) and the fifth is obtained by using (L2), (L3) and (L5) like in case (**ii**) above.

ii. *l' is extruded:* in this case, $\sigma' \triangleq \sigma_1 \cdot (\nu l')\,\langle l'\rangle\ @\ l''\ \cdot \sigma_2 \cdot$ **nil** $@\ l\ \cdot \sigma_3$ and $\amalg(\sigma'') \xRightarrow{\sigma_1 \cdot\ \langle l'\rangle\ @\ l''\ \cdot \sigma_2 \cdot \sigma_3' \cdot OK}$, where σ_3' has been obtained from σ_3 like in (**ii**) before. By induction, prefix closure and by following steps similar to case 2.(a).i, we can prove that $\sigma \geq \phi \cdot \overline{\sigma_1} \cdot\ l' \triangleleft l''\ \cdot \overline{\sigma_2} \cdot \overline{\sigma_3'} \geq \overline{\sigma_1} \cdot (\nu l')\, l' \triangleleft l''\ \cdot \overline{\sigma_2} \cdot \triangleright l \cdot \overline{\sigma_3} \triangleq \overline{\sigma'}$. The only difference with the previous inference is that the $(\nu l')$ at the beginning of ϕ is not thrown away but is captured by $l' \triangleleft l''$, as desirable.
Remark: it could also be $\sigma' \triangleq \sigma_1 \cdot$ **nil** $@\ l\ \cdot \sigma_2 \cdot (\nu l')\,\langle l'\rangle\ @\ l''\ \cdot \sigma_3$. This case can be easily adapted, by considering $\amalg(\sigma'') \xRightarrow{\sigma_1 \cdot \sigma_2' \cdot\ \langle l'\rangle\ @\ l''\ \cdot \sigma_3' \cdot OK}$, where σ_i' has been obtained from σ_i like in (**ii**).

(b) *it also offers the datum via a label $(\nu l')\,\langle l'\rangle\ @\ l$ (that also extrudes l'):* in this case, $\sigma' \triangleq \sigma_1 \cdot$ **nil** $@\ l\ \cdot \sigma_2 \cdot (\nu l')\,\langle l'\rangle\ @\ l\ \cdot \sigma_3$ and $\amalg(\sigma'') \xRightarrow{\sigma_1 \cdot \sigma_2' \cdot \sigma_3' \cdot OK}$, where σ_2' and σ_3' have been obtained from σ_2 and σ_3 like in (**ii**) above. Then, $\sigma \geq \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2'} \cdot \overline{\sigma_3'} \geq\ \triangleright l\ \cdot \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2'} \cdot \overline{\sigma_3'} \geq \overline{\sigma_1} \cdot\ \triangleright l\ \cdot \overline{\sigma_2} \cdot \phi \cdot \overline{\sigma_3} \triangleq \overline{\sigma'}$, where we have used laws (L5) and (L2), possibly iterated several times, and law (L3) if needed.

(c) *it also offers the datum via a label $\langle l'\rangle\ @\ l$ but l' has been previously extruded:* in this case, $\sigma' \triangleq \sigma_1 \cdot$ **nil** $@\ l\ \cdot \sigma_2 \cdot (\nu l')\,\langle l'\rangle\ @\ l''\ \cdot \sigma_3 \cdot\ \langle l'\rangle\ @\ l\ \cdot \sigma_4$ and $\amalg(\sigma'') \xRightarrow{\sigma_1 \cdot \sigma_2' \cdot\ \langle l'\rangle\ @\ l''\ \cdot \sigma_3' \cdot \sigma_4' \cdot OK}$, where the σ_i's have been obtained from the corresponding σ_is like in (**ii**) above. The proof proceeds like in the previous cases, by using (L5), (L2) and (L3). Again, like in case 2.(a).ii, the restriction on l' is captured by $l' \triangleleft l''$, as required. Moreover, like in case 2.(a).ii, the proof is not radically changed if we consider $\sigma' \triangleq \sigma_1 \cdot (\nu l')\,\langle l'\rangle\ @\ l''\ \cdot \sigma_2 \cdot$ **nil** $@\ l\ \cdot \sigma_3 \cdot\ \langle l'\rangle\ @\ l\ \cdot \sigma_4$.

(d) *the datum $\langle l'\rangle$ is then passed to C*[`test` :: *P*] *with a communication*

i. *l' is not extruded:* in this case, $\sigma' \triangleq \sigma_1 \cdot$ **nil** $@\ l\ \cdot \sigma_2 \cdot \sigma_3$ and $\amalg(\sigma'') \xRightarrow{\sigma_1 \cdot \sigma_2' \cdot\ l' \triangleleft l\ \cdot \sigma_3' \cdot OK}$, where σ_2' and σ_3' have been obtained from σ_2 and σ_3 like in (**ii**) above. Then, $\sigma \geq \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2'} \cdot \langle l'\rangle\ @\ l \cdot \overline{\sigma_3'} \geq\ \triangleright l \cdot \phi \cdot \overline{\sigma_1} \cdot \overline{\sigma_2'} \cdot \langle l'\rangle\ @\ l \cdot \overline{\sigma_3'} \geq \overline{\sigma_1} \cdot\ \triangleright l\ \cdot \overline{\sigma_2} \cdot \phi \cdot\ \langle l'\rangle\ @\ l\ \cdot \overline{\sigma_3} \geq \overline{\sigma_1} \cdot\ \triangleright l\ \cdot \overline{\sigma_2} \cdot \overline{\sigma_3} \triangleq \overline{\sigma'}$, where the second step relies on law (L5), the third step on laws (L2), (L5) and (L3) used as needed, and the fourth step relies on rule (L3). Notice that, since l' is not extruded, it must be that $l' \notin fn(\sigma_3)$; thus, $(\nu l')\overline{\sigma_3} = \overline{\sigma_3}$.
Remark: if l' is extruded after the communication, σ_3 will be of the form

$\ldots \cdot \langle l' \rangle\ @\ l'' \cdot \ldots$; hence, $\overline{\sigma_3}$ is $\overline{\ldots} \cdot\ l' \triangleleft l'' \cdot \overline{\ldots}$. Now, $(\nu l')\overline{\sigma_3} = \overline{\ldots} \cdot (\nu l')\ l' \triangleleft l'' \cdot \overline{\ldots}$ and the proof carries on in the same way.

ii. *l' has been previously extruded:* in this case, $\sigma' \triangleq \sigma_1 \cdot$ **nil** $@\ l\ \cdot \sigma_2 \cdot (\nu l')\ \langle l' \rangle\ @\ l''\ \cdot \sigma_3 \cdot \sigma_4$ and $\text{Ш}(\sigma'') \xRightarrow{\sigma_1 \cdot \sigma'_2 \cdot\ \langle l' \rangle\ @\ l''\ \cdot \sigma'_3 \cdot\ l' \triangleleft l\ \cdot \sigma'_4 \cdot OK}$, where the σ'_is have been obtained from the corresponding σ_is like in (**ii**) above. The proof is carried on similarly to the previous cases. The situation in which the extrusion of l' proceeds the label **nil** $@\ l$ is similar.

3. *the first contribution of $l :: \langle l' \rangle$ in σ' is with $(\nu l')\ \langle l' \rangle\ @\ l$ (that also extrudes l'):* in this case, $\sigma' \triangleq \sigma_1 \cdot (\nu l')\ \langle l' \rangle\ @\ l\ \cdot \sigma_2$ and $\text{Ш}(\sigma'') \xRightarrow{\sigma_1 \cdot \sigma'_2 \cdot OK}$, where σ'_2 has been obtained from σ_2 like in (**ii**). The thesis follows by induction, prefix closure, law (L2) and by possibly repeated applications of laws (L5), (L2) and (L3).
4. *the first contribution of $l :: \langle l' \rangle$ in σ' is with $\langle l' \rangle\ @\ l$ and l' has been previously extruded:* in this case, $\sigma' \triangleq \sigma_1 \cdot (\nu l')\ \langle l' \rangle\ @\ l''\ \cdot \sigma_2 \cdot\ \langle l' \rangle\ @\ l\ \cdot \sigma_3$ and $\text{Ш}(\sigma'') \xRightarrow{\sigma_1 \cdot\ \langle l' \rangle\ @\ l''\ \cdot \sigma_2 \cdot \sigma'_3 \cdot OK}$, where σ'_3 has been obtained from σ_3 like in (**ii**). The situation is like in the previous case, but now the restriction on l' remains associated to $l' \triangleleft l''$, as needed.
5. *the first contribution of $l :: \langle l' \rangle$ to the production of σ' is by passing the datum to C*[`test` :: *P*] *with a communication*
   (a) *l' is not extruded:* this case is similar to 2.(d).i above, but it is simpler. Indeed, $\sigma' \triangleq \sigma_1 \cdot \sigma_2$ and $\text{Ш}(\sigma'') \xRightarrow{\sigma_1 \cdot\ l' \triangleleft l\ \cdot \sigma'_2 \cdot OK}$. The situation does not radically change if l' is extruded after the communication (see the Remark at the end of case 2.(d).i above).
   (b) *l' has been previously extruded:* this case is similar to 2.(d).ii above. Indeed, $\sigma' \triangleq \sigma_1 \cdot (\nu l')\ \langle l' \rangle\ @\ l''\ \cdot \sigma_2 \cdot \sigma_3$ and $\text{Ш}(\sigma'') \xRightarrow{\sigma_1 \cdot\ \langle l' \rangle\ @\ l''\ \cdot \sigma_2 \cdot\ l' \triangleleft l\ \cdot \sigma'_3 \cdot OK}$. ∎

Finally, we can prove that trace equivalence is a sound proof technique for may testing (see Theorem 5.4) that exactly captures it.

Theorem 5.8 (Completeness of $\asymp$ w.r.t. $\simeq$) *If $N \simeq M$ then $N \asymp M$.*

Proof: Let σ be a trace of N, i.e. $N \xRightarrow{\sigma}$. By Proposition 5.5, $\text{Ш}(\sigma) \xRightarrow{\overline{\sigma} \cdot OK}$; thus, by Lemma 5.2.1, $N \parallel \text{Ш}(\sigma) \xRightarrow{OK}$. By Proposition 3.5 and Definition 3.4, it holds that $M \parallel \text{Ш}(\sigma) \xRightarrow{OK}$. By Lemma 5.6, there exists σ' such that $M \xRightarrow{\sigma'}$, $\text{Ш}(\sigma) \xRightarrow{\overline{\sigma'} \cdot OK}$ and σ' does not contain labels of the form $\trianglerighteq l$. By Lemma 5.7 (notice that, since `test` is fresh, it holds that `test` $\notin n(\sigma')$), $\sigma' \preceq \sigma$ as required by Definition 5.1; thus, $N \asymp M$. ∎

Corollary 5.9 (Tractable Characterization of May Testing) $\asymp\ =\ \simeq$.

6 Verifying a Distributed Protocol: The Dining Philosophers

We now use the proof techniques we have just presented to state and prove the properties of a ‘classical’ problem in distributed systems, namely the *‘Dining Philosophers’*.[4] In what follows,

[4]Historically, the problem was first formulated and solved by Dijkstra in 1965 and was used to motivate the use of semaphores.

we shall use bisimulation, that is finer but easier to prove. All the work can be done with trace equivalence as well.

To have a more elegant implementation of the protocol and an easier verification, in this section we shall use polyadic data, i.e. we shall consider data of the form $\langle l_1, \ldots, l_n \rangle$. In [21] we prove that this feature does not radically improve the expressive power of the calculus. By using some terminology from LINDA [26], we use the following extended syntax of cKLAIM:

$$\begin{array}{llcl}
\textit{Tuples} & t & ::= & u \;\big|\; t_1, t_2 \\
\textit{Templates} & T & ::= & p \;\big|\; T_1, T_2 \\
\textit{Actions} & a & ::= & \mathbf{out}(t)@u \;\big|\; \mathbf{in}(T)@u \;\big|\; \mathbf{eval}(P)@u \;\big|\; \mathbf{new}(l)
\end{array}$$

The remaining productions are like in Table 1. Now, a tuple $\langle t \rangle$ can be retrieved by means of a template T if they both have the same number of fields and corresponding fields match (i.e., a bound variable matches any name, while two names match only if identical). In this case, we write $match(T, t)$. Now, rules (LTS-IN) and (LTS-MATCH) are joint together in rule

$$\frac{match(T, t)}{l :: \mathbf{in}(T)@l'.P \xrightarrow{t \triangleleft l'} l :: P[{}^{t}\!/\!_{T}]}$$

Process $P[{}^{t}\!/\!_{T}]$ is obtained from P by replacing all the occurrences of variables bound in T with the corresponding name in t. A similar adaption is needed also for rules (R-IN) and (R-MATCH). We let $\approx_p$ and $\cong_p$ be the bisimulation and the barbed congruence in the polyadic case. It is trivial to prove that $\approx_p = \cong_p$; thus, in what follows, we are justified to use $\approx_p$.

The problem. The dining philosophers is a "classical" synchronisation problem; its luck derives from the fact that it naturally models many synchronisation problems arising when allocating resources in concurrent/distributed systems. The problem can be described as follows. Some, say n, philosophers spend their lives alternating between thinking and eating. They are seated around a circular table and there is a fork placed between each pair of neighbouring philosophers. Each philosopher has access to the forks at his left and right; if a philosopher wants to eat, he has to acquire both the forks near to him (this is possible only if none of his neighbours are using the forks); when done eating, the philosopher puts both forks back down on the table and begins thinking. The challenge in the dining philosophers problem is to design a protocol so that the philosophers do not deadlock (i.e. the entire set of philosophers does not stop and wait indefinitely), and so that no philosopher starves (i.e. each philosopher eventually gets his hands on a pair of forks). Additionally, the protocol should be as efficient as possible – in other words, the time that philosophers spend waiting to eat should be minimised.

Our solution. We now propose a protocol in μKLAIM to solve the problem, in the same spirit as Dijkstra's solution. We shall associate each philosopher with a distinct locality taken from the set $\{l_1, \ldots, l_n\}$. We also use a restricted locality l to record in a tuple of length n the allocation of the forks (i.e. the status of each philosopher); more precisely, if the i-th component of this tuple is `t` then the i-th philosopher is thinking, if it is `e` the i-th philosopher is eating. The access to such a tuple will allow the processes to act on resources allocation in mutual exclusion. The node l_i will then host the following process (implementing the behaviour of the i-th philosopher):

$$\mathbf{rec}\, X.\ think\,.\,\mathbf{in}(T_i)@l.\mathbf{out}(t_i)@l\,.\,eat\,.\,\mathbf{in}(T_i')@l.\mathbf{out}(t_i')@l.X$$

where

$$
\begin{array}{rcll}
T_i & \triangleq & \left\{ \begin{array}{l} \mathtt{t}, \mathtt{t}, !x_3, \ldots, !x_{n-1}, \mathtt{t} \\ !x_1, \ldots, !x_{i-2}, \mathtt{t}, \mathtt{t}, \mathtt{t}, !x_{i+2}, \ldots, !x_n \\ \mathtt{t}, !x_2, \ldots, !x_{n-2}, \mathtt{t}, \mathtt{t} \end{array} \right. & \begin{array}{l} \text{if } i = 1 \\ \text{if } 1 < i < n \\ \text{if } i = n \end{array} \\
t_i & \triangleq & \left\{ \begin{array}{l} \mathtt{e}, \mathtt{t}, x_3, \ldots, x_{n-1}, \mathtt{t} \\ x_1, \ldots, x_{i-2}, \mathtt{t}, \mathtt{e}, \mathtt{t}, x_{i+2}, \ldots, x_n \\ \mathtt{t}, x_2, \ldots, x_{n-2}, \mathtt{t}, \mathtt{e} \end{array} \right. & \begin{array}{l} \text{if } i = 1 \\ \text{if } 1 < i < n \\ \text{if } i = n \end{array} \\
T'_i & \triangleq & \left\{ \begin{array}{l} \mathtt{e}, \mathtt{t}, !y_3, \ldots, !y_{n-1}, \mathtt{t} \\ !y_1, \ldots, !y_{i-2}, \mathtt{t}, \mathtt{e}, \mathtt{t}, !y_{i+2}, \ldots, !y_n \\ \mathtt{t}, !y_2, \ldots, !y_{n-2}, \mathtt{t}, \mathtt{e} \end{array} \right. & \begin{array}{l} \text{if } i = 1 \\ \text{if } 1 < i < n \\ \text{if } i = n \end{array} \\
t'_i & \triangleq & \left\{ \begin{array}{l} \mathtt{t}, \mathtt{t}, y_3, \ldots, y_{n-1}, \mathtt{t} \\ y_1, \ldots, y_{i-2}, \mathtt{t}, \mathtt{t}, \mathtt{t}, y_{i+2}, \ldots, y_n \\ \mathtt{t}, y_2, \ldots, y_{n-2}, \mathtt{t}, \mathtt{t} \end{array} \right. & \begin{array}{l} \text{if } i = 1 \\ \text{if } 1 < i < n \\ \text{if } i = n \end{array}
\end{array}
$$

Intuitively, the first **in** action verifies that the neighbors of the *i*-th philosopher are not eating and simoultaneously acquires the lock on the tuple; then the **out** action sets the status of the *i*-th philosopher to e, while releasing the lock. Then, the following **in** and **out** actions release the resources used upon completion of the eating phase and the protocol iterates.

For the sake of simplicity, we do not model the *think* phase, while the *eat* phase is just an **out** action over some (fresh) locality l'. Hence, if the system starts with all the philosophers in a thinking state, the net implementing the system is

$$
\begin{array}{rcl}
N & \triangleq & (\nu l)(\, l :: \langle \mathtt{t}, \ldots, \mathtt{t} \rangle \;\|\; \prod_{i=1}^{n} l_i :: P_i \,) \\
P_i & \triangleq & \mathbf{rec}\, X.\mathbf{in}(T_i)@l.\mathbf{out}(t_i)@l.\mathbf{out}(l_i)@l'.\mathbf{in}(T'_i)@l.\mathbf{out}(t'_i)@l.X
\end{array}
$$

Soundness of our solution. We shall now verify the correctness of our protocol, namely that (1) no deadlock nor starvation ever occur, (2) resources are properly used (namely, no neighboring philosophers eat at the same time) and (3) the protocol enables the highest level of parallelism (i.e. it is possible for $\lfloor \frac{n}{2} \rfloor$ philosophers to eat together).

1. We shall prove that

$$N \;\|\; l' :: \mathbf{nil} \;\approx_p\; N \;\|\; l' :: \langle l_i \rangle \qquad (1)$$

for each $i = 1, \ldots, n$. Equation (1) can be proved by showing that the relations $\Re_1^i \triangleq \{ (N', N' \;\|\; l' :: \langle l_i \rangle) \;:\; N \;\|\; l' :: \mathbf{nil} \Rightarrow N' \} \cup Id$ are weak bisimulations (up-to $\equiv$); this can be done easily. This means that computations from N can never get stuck (hence deadlock will never occur) and that each philosopher can eat an unbounded number of times (hence starvation cannot occur).

Deadlock freedom: To prove it, we proceed by contradiction; hence, let us suppose that there exists a computation from N leading to deadlock. Since the computation is finite, we can find an integer k which is an upper bound to the number of steps performed by N before reaching the deadlock. But then, we can iterate $k + 1$ times Equation (1) and (the polyadic case of) Theorem 4.7 to obtain that $N \;\|\; l' :: \mathbf{nil} \;\approx_p\; N \;\|\; \prod_{j=1}^{k+1} l' :: \langle l_i \rangle$. This equivalence is however contradicted by letting $N \;\|\; l' :: \mathbf{nil}$ to follow the computation leading to deadlock. Indeed, since $N \;\|\; l' :: \mathbf{nil}$ performs at most k steps in such computation, it is impossible for it to produce $k + 1$ data in l' (recall that l' is fresh for N); on the other hand, no computation from $N \;\|\; \prod_{j=1}^{k+1} l' :: \langle l_i \rangle$ will ever remove data from l' (because l' has been chosen fresh for N). Thus, the resulting nets exhibit different data in l' and cannot be equivalent.

Starvation freedom: The proof is similar. Indeed, if there exists a computation from N starving philosopher i by letting him to eat at most k times, then such a computation contradicts $N \parallel l' :: \mathbf{nil} \ \approx_p \ N \parallel \prod_{j=1}^{k+1} l' :: \langle l_i \rangle$.

2. Let l'' be a fresh locality. We define

$$\begin{array}{lcl} M & \triangleq & l :: \langle \mathtt{t}, \ldots, \mathtt{t} \rangle \parallel \prod_{i=1}^{n} l_i :: P_i \\ C[\cdot] & \triangleq & l'' :: \mathbf{nil} \parallel (\nu l, l')[\cdot] \\ \mathcal{D}[\cdot] & \triangleq & l'' :: \mathbf{nil} \parallel (\nu l, l')\big([\cdot] \parallel l :: \ \mathbf{in}(\mathtt{e}, \mathtt{e}, !x_3, \ldots, !x_n)@l.\mathbf{out}()@l'' \\ & & \qquad\qquad\qquad\qquad \mid \ \mathbf{in}(!x_1, \mathtt{e}, \mathtt{e}, !x_4, \ldots, !x_n)@l.\mathbf{out}()@l'' \mid \ldots \\ & & \qquad\qquad\qquad\qquad \mid \ \mathbf{in}(\mathtt{e}, !x_2, \ldots, !x_{n-1}, \mathtt{e})@l.\mathbf{out}()@l'' \big) \end{array}$$

Notice that $N \triangleq (\nu l)M$ and hence $C[M] \triangleq l'' :: \mathbf{nil} \parallel (\nu l')N$. We have restricted node l' because we are not interested in observing who is eating (and because this simplifies the formulation of Equation (2) below); we later show that this fact implies that N must access resources properly. We want to prove that

$$C[M] \ \approx_p \ \mathcal{D}[M] \tag{2}$$

i.e. $\mathcal{D}[M]$ will never produce data at l'' (since $C[M]$ cannot). Intuitively, $\mathcal{D}[M]$ can produce a datum at l'' if it happens that two adiacent philosophers eat simoultaneously; hence Equation (2) implies that no resource is ever misused. The above equation can be proved by showing that the relation $\Re_2 \triangleq \{(C[M'], \mathcal{D}[M']) \ : \ C[M] \Rightarrow C[M']\}$ is a bisimulation; again, this is an easy task.

Now, suppose that there exists a computation from N misusing the resources; this means that $N \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_k} N'$ and N' is a net where two adjacent philosophers are eating simoultaneously. Thus $N' \triangleq (\nu l)(l :: \langle \ldots, \mathtt{e}, \mathtt{e}, \ldots \rangle \ \parallel \ \ldots)$ where the two $\mathtt{e}$ are adjacent modulo n. But then $\mathcal{D}[M] \xrightarrow{\alpha'_1} \ldots \xrightarrow{\alpha'_k} \mathcal{D}[M']$ where $N' \triangleq (\nu l)M'$, $\alpha'_i = \alpha_i$ if l' is not the target of α_i, and $\alpha'_i = \tau$ otherwise. Hence, $\mathcal{D}[M'] \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\langle\rangle @ l''}$, thus contradicting Equation (2).

3. The easiest way to prove that $\lfloor \frac{n}{2} \rfloor$ philosophers can eat simultaneously is to show a computation from N leading to a tuple in l with exactly $\lfloor \frac{n}{2} \rfloor$ items of kind $\mathtt{e}$, while respecting the correct use of resources. The wanted reduction is obtained by letting the even philosophers accessing in turn the status tuple. This is always possible since an even philosopher is always surrounded (modulo n) by two, not eating, odd philosophers. Hence we have that

$$\begin{array}{lll} N & \xrightarrow{\tau}\xrightarrow{\tau} & l :: \langle \mathtt{t}, \mathtt{e}, \mathtt{t}, \mathtt{t}, \ldots, \mathtt{t} \rangle \parallel l_2 :: \mathbf{out}(l_2)@l'.\mathbf{in}(T'_2)@l.\mathbf{out}(t'_2)@l.P_2 \parallel \prod_{i=1,\ldots,n}^{i\neq 2} l_i :: P_i \\ & \xrightarrow{\tau}\xrightarrow{\tau} & l :: \langle \mathtt{t}, \mathtt{e}, \mathtt{t}, \mathtt{e}, \mathtt{t}, \mathtt{t}, \ldots, \mathtt{t} \rangle \parallel l_2 :: \mathbf{out}(l_2)@l'.\mathbf{in}(T'_2)@l.\mathbf{out}(t'_2)@l.P_2 \parallel \\ & & l_4 :: \mathbf{out}(l_4)@l'.\mathbf{in}(T'_4)@l.\mathbf{out}(t'_4)@l.P_4 \parallel \prod_{i=1,\ldots,n}^{i\neq 2,4} l_i :: P_i \\ & \ldots & \\ & \xrightarrow{\tau}\xrightarrow{\tau} & l :: \langle \mathtt{t}, \mathtt{e}, \mathtt{t}, \mathtt{e}, \ldots \rangle \parallel \prod_{i=1,\ldots,n}^{i \text{ even}} l_i :: \mathbf{out}(l_i)@l'.\mathbf{in}(T'_i)@l.\mathbf{out}(t'_i)@l.P_i \parallel \prod_{i=1,\ldots,n}^{i \text{ odd}} l_i :: P_i \end{array}$$

7 Equational Laws and the Impact of Richer Contexts

In this section, we want to discuss some equational laws that can be easily proved by exploiting both bisimulation and trace equivalence. We concentrate on bisimulation that is finer (by virtue of Proposition 3.5 and Theorems 4.8 and 5.8). The first law is inspired from the asynchronous π-calculus [4]

$$l' :: \mathbf{rec}\, X.\mathbf{in}(!x)@l.\mathbf{out}(x)@l.X \;\approx\; l' :: \mathbf{nil}$$

and states that (repeatedly) accessing a datum and putting it back in its original location is observationally equivalent to performing no operation. Of course, this heavily exploits the fact that communication in cKlaim is asynchronous. This law also motivates the choice to omit from cKlaim the X-Klaim action **read**. In fact, action **read** is relevant, e.g., for security reasons (removing a datum while accessing it via an **in** requires a different capability than simply accessing it via a **read**) that are ignored in this paper.

We have also the following four significant laws (the last one can be easily derived from the second and the third one):

$$l :: \mathbf{out}(l'')@l'.P \parallel l' :: \mathbf{nil} \quad\approx\quad l :: P \parallel l' :: \langle l''\rangle \tag{3}$$
$$l :: \mathbf{eval}(Q)@l'.P \parallel l' :: \mathbf{nil} \quad\approx\quad l :: P \parallel l' :: Q \tag{4}$$
$$l :: P|Q \parallel l' :: \mathbf{nil} \quad\approx\quad l :: P \parallel l' :: Q \tag{5}$$
$$l :: \mathbf{eval}(Q)@l'.P \parallel l' :: \mathbf{nil} \quad\approx\quad l :: P|Q \parallel l' :: \mathbf{nil} \tag{6}$$

Laws 3 and 4 state that it is impossible to know when data and processes have been allocated – either at the outset or during computations. Law 5 states that, once the net is fixed, the actual distribution of processes is irrelevant, while law 6 states that remotely executing a process is observationally equivalent to executing the process locally. At a first sight, these laws could be quite surprising and seem to contradict the design principles at the basis of cKlaim. However, they can be explained by observing the net at a very high level, namely at the level of the user applications. Indeed, we are observing the functionalities a net offers to a terminal user. Therefore, the allocation of processes cannot be observed (law 5) and the advantages of exploiting mobile processes (e.g. efficiency, reduced network load, support for disconnected operations) cannot be perceived at all (law 6).

In many circumstances this level of abstraction is exactly what we need. For example, when we studied the 'Dining philosophers', we were interested in the overall behaviour of the system and in the properties it enjoyed; thus, we could ignore the implementation details and take into account only the functional aspects of the protocol. If we want more details on the distributed environment underlying a cKlaim application, we have to refine the observation level. Consequently, to study lower-level aspects like, e.g., routing problems or failures, we have to adapt the language and the semantic theories we developed in this paper. To this aim, we have studied three variants of cKlaim where (*i*) communication can only take place locally, (*ii*) failures (of both components and nodes) can occur, and (*iii*) dynamically evolving connections between nodes are explicitly modelled. Later on, we shall give some hints on the first two variants and leave the more elaborated treatment of the third scenario for a companion paper [20]. Predictably, laws 5 and 6 do not hold in these lower-level settings.

Local Communications. We start by modifying the syntax of Table 1 in order to forbid remote executions of actions **in** and **out**. The productions for process actions now become

$$a \;::=\; \mathbf{in}(p) \;\Big|\; \mathbf{out}(u) \;\Big|\; \mathbf{eval}(Q)@u \;\Big|\; \mathbf{new}(l)$$

Rules (R-Out), (R-In) and (R-Match) are modified accordingly to become

$$\begin{array}{lrcl}
\text{(R-Out)} & l :: \mathbf{out}(l').P & \longmapsto & l :: P\,|\,\langle l'\rangle \\
\text{(R-In)} & l :: \mathbf{in}(!x).P\,|\,\langle l'\rangle & \longmapsto & l :: P[{}^{l'}\!/_{x}] \\
\text{(R-Match)} & l :: \mathbf{in}(l').P\,|\,\langle l'\rangle & \longmapsto & l :: P
\end{array}$$

Let $\cong_l$ and $\simeq_l$ be the barbed congruence and the may testing in the resulting calculus. We now show that the proof techniques developed in Sections 4 and 5 still hold in this more localised framework. To this aim, we need to modify the LTS of Table 5; now, rules (LTS-OUT), (LTS-IN) and (LTS-MATCH) become

$$\begin{array}{ll}
\text{(LTS-OUT)} & l :: \mathbf{out}(l').P \xrightarrow{\triangleright\, l} l :: P \,|\, \langle l' \rangle \\
\text{(LTS-IN)} & l :: \mathbf{in}(!x).P \xrightarrow{l' \,\triangleleft\, l} l :: P[^{l'}\!/_{x}] \\
\text{(LTS-MATCH)} & l :: \mathbf{in}(l').P \xrightarrow{l' \,\triangleleft\, l} l :: P
\end{array}$$

By letting $\approx_l$ and $\asymp_l$ be the bisimulation and testing equivalences defined like in Sections 4 and 5 on top of this modified LTS, we can prove the analogous of our main results.

Theorem 7.1 $\approx_l \;=\; \cong_l \;\subset\; \simeq_l \;=\; \asymp_l$.

Proof: The proofs of the three claims can be easily adapted from those of Theorems 4.8, 4.12, 5.4 and 5.8, and by exploiting Proposition 3.5. ∎

It is very easy to check that

$$l :: P \parallel l' :: \mathbf{nil} \not\simeq_l l :: \mathbf{nil} \parallel l' :: P \qquad l :: \mathbf{eval}(Q)@l'.P \parallel l' :: \mathbf{nil} \not\simeq_l l :: P|Q \parallel l' :: \mathbf{nil}$$

This is reasonable because, since communications are local, by moving a process we also change its execution environment. Thus, at the very least, its observable behaviour will change according to the node where it runs. Notice that, in order to disprove laws 5 and 6 we have used may testing. Indeed, because of Theorem 7.1, $\not\simeq_l$ implies $\not\cong_l$.

Failures. Now, we consider another setting and enrich cKLAIM with a mechanism for modelling various forms of failures. This is achieved by adding the following rules to the definition of the reduction relation and of the LTS:

$$\text{(R-FAIL)} \quad l :: C \longmapsto \mathbf{0} \qquad\qquad \text{(LTS-FAIL)} \quad l :: C \xrightarrow{\tau} \mathbf{0}$$

These rules model corruption of data (*message omission*) if $C \triangleq \langle d_1 \rangle | \ldots | \langle d_n \rangle$, node (*fail-silent*) failure if $l :: C$ collects all the components located at l, and abnormal termination of some processes running at l otherwise. In this way, we model failures as disappearance of a resource (a datum, a process or a whole node). This is a simple, but realistic, way of representing failures, specifically fail-silent and message omission, in a global computing scenario [12]. Indeed, while the presence of data/nodes can be ascertained, their absence cannot because in such a scenario there is no practical upper bound to communication delays. Thus, failures cannot be distinguished from long delays and should be modelled as totally asynchronous and undetectable events.

Again, it is easy to prove that laws 5 and 6 given for cKLAIM do not hold anymore in this more concrete setting. Indeed, the failure of l' can easily modify the overall behaviour of the equated nets. We now examine what happens to the characterization of barbed congruence and may testing in this new framework. The definition of the bisimulation equivalence does not need to be modified to exactly capture barbed congruence. Indeed, the recursive closure of both barbed congruence and bisimulation already forces the corruption of the same data and the failure of the same nodes to take place at the same time; as regards process abnormal termination, it will be the evolution of the involved nets that will affect the equivalence. About trace equivalence, the characterization breaks down: trace equivalence is only a sound (but not complete) proof technique for may testing. The problem is that Lemmas 5.6 and 5.7 do not hold anymore in the lower-level setting. This does not mean that trace equivalence is strictly finer than may testing, even if we believe this; it only means that the proof of Theorem 5.8 must be carefully re-examined. A more precise statement on this aspect is left for future work.

If we let $\approx_f$, $\cong_f$, $\asymp_f$ and $\simeq_f$ denote labelled bisimilarity, barbed congruence, trace equivalence and may testing for the calculus with failures, we have

Theorem 7.2 $\approx_f \ = \ \cong_f$ *and* $\asymp_f \ \subseteq \ \simeq_f$.

Proof: The proof is formally identical to those of Theorems 4.8, 4.12 and 5.4, but now a τ-step or a reduction can be also generated by applying rule (LTS-FAIL) and (R-FAIL) respectively. ∎

8 Conclusions and Related Work

We have presented some semantic theories for cKLAIM, a process calculus with process distribution, process mobility, remote operations and asynchronous communication through distributed repositories. This combination of design choices has already proved to be valuable from an implementative and applicative point of view. The semantic theories we introduced in this paper have been defined in a uniform fashion [10]: first, we defined some user basic observables for a global computing setting; then, we closed them under all possible contexts and/or reductions, thus obtaining two touchstone equivalences (namely *barbed congruence* and *may testing*); finally, we gave tractable characterisations of these equivalences by means of *labelled bisimulation* and *trace equivalence*. We have also discussed if and how these theories change when extending cKLAIM with lower-level mechanisms like modelling of failures and implementing remote communications via migrations and local exchanges.

Future work. Possible developments of this work include the study of abstractions, e.g. administrative domains and security policies, that determine *virtual* networks on top of the effective one. To this aim, dynamically evolving type environments could be exploited to constraint the behaviours of processes and the observations of an environment. Some work in this direction has been done in [28].

Finally, it would also be interesting to analyze efficiency issues to better clarify, e.g., the advantages of mobile code and process distribution. A possible application of laws 5 and 6 in Section 7 is to find out possible rearrangements of the processes over a given net that minimize the number of remote operations. In fact, it is reasonable to assume that local operations are cheaper and faster than remote ones. Thus, we can re-locate the parallel components or spawn some of the processes running at some locality to improve the overall net behaviour.

Related work. We conclude by reviewing related work on observational equivalences for calculi with process distribution and mobility (many of them are surveyed in [15]). In the nineties, many CCS-like process calculi have been enriched with localities to explicitly describe the distribution of processes. The aim was mainly to provide these calculi with non interleaving semantics or, at least, to differentiate processes' parallel components (thus obtaining more inspective semantics than the interleaving ones). This line of research is far from the one in which cKLAIM falls, where localities are used as a mean to make processes network aware thus enabling them to refer to the network locations as target of remote communication or as destination of migrations. Localities are not only considered as units of distribution but, according to the case, as units of mobility, of communication, of failure or of security.

[39] and [3] extend, resp., CCS and π-calculus with process distribution and mobility. In both cases, processes run over the nodes of an explicit, flat and dynamically evolving net architecture. Nodes can fail thus causing loss of all hosted processes. There are explicit operations to kill nodes and to query the status of a node. Failures can be detected, which is suitable for distributed computing but clashes with the assumptions underlying global computing. In both papers, a labelled bisimulation (akin to the bisimulation in the CCS and π-calculus) is given to capture a standardly defined barbed congruence.

Another distributed version of the π-calculus is presented in [28]; the resulting calculus contains primitives for code movement and creation of new localities/channels in a net with a flat architecture. Over the LTS defining the semantics of the calculus, a typed bisimulation (with a tractable formulation) is defined that exactly capture typed barbed equivalence. The use of types illustrates the importance of having the rights to observe a given behaviour: indeed, different typings (i.e. observation rights) generate different bisimulations, that are finer as long as the typing is less restrictive.

In the Distributed Join calculus [24], located mobile processes are hierarchically structured and form a tree-like structure evolving during the computation. Entire subtrees, not only single processes, can move). Technically, nets are flat collections of named nodes, where the name of a node indicates the nesting path of the node; e.g., a node whose name is $l_1.\cdots.l_k.l$ represents a node referrable to via the unique name l and that is contained in l_k, that is a node contained in l_{k-1} and so on. Communication in DJoin takes place in two steps: firstly, the sending process sends a message on a channel; then, the ether (i.e. the environment containing all the nodes) delivers the message to the (unique) process that can receive on that channel. The fact that in the whole net there is a unique process capable to receive at a given channel makes DJoin communication somehow similar to cKlaim one, in that DJoin channels have a role similar to that of cKlaim localities. Failures are modelled by tagging locality names: e.g. the (compound) name $\cdots.l_i^{\Omega}.\cdots.l$ states that l is a node contained in a failed node l_i and, thus, l itself is failed. The Ω at l_i has been caused by execution of the primitive *halt* by a process running at l_i. Failures can be detected by using the primitive $fail$. Failed nodes cannot host running computations but can receive data/code/sublocations that, however, once arrived in the failed node, become definitely stuck. Some interesting laws and properties are proved using a contextual barbed equivalence, but no tractable characterization of the equivalence is given and it is not even obvious how to extend the characterization of barbed bisimulation for the (non-distributed) Join calculus introduced in [25] to account for distribution and agent mobility.

The Ambient calculus [13] is an elegant notation to model hierarchically structured distributed applications. Though the definition of its reduction semantics is very simple, the formulation of a reasonable, possibly tractable, observational equivalence is a very hard task. The calculus is centered around the notion of connections between ambients, that are containers of processes and data. Each primitive can be executed only if the ambient hierarchy is structured in a precise way; e.g., an ambient n can enter an ambient m only if n and m are sibling, i.e. they are both contained in the same ambient. This fact greatly complicates the definition of a tractable equivalence. Recently, in [31], a bisimulation capturing Ambient's barbed congruence has been defined. This has been done by structuring the syntax into two levels, namely processes and nets (where the latter ones are particular cases of the former ones), and by exploiting an involved LTS (using three different kinds of labels some of which containing process contexts). However, the defined bisimulation is not standard and suffers from a quantification over all the possible processes (to fill in the 'holes' generated by the operational semantics).

Similar bisimulations have also been developed for calculi derived from Ambient, like, e.g., Safe Ambients [30], Boxed Ambients [11], the Seal Calculus [14] and the calculus of Mobile Resources [27]. Moreover, in the last three papers, bisimulation is only a sound but not complete proof technique for barbed congruence.

To conclude, we want to remark that, to the best of our knowledge, no characterization of may testing in terms of trace equivalence has even been given for an asynchronous, distributed language with process mobility. In [42], a theory for may testing (and the corresponding characterization) is developed for the Actors Model [1]. However, the work is done by reducing Actors to a typed asynchronous π-calculus and the trace-based characterisation follows [9].

References

[1] G. Agha. *A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[2] M. Agrawal and A. Seth, editors. *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference Kanpur, India, December 12-14, 2002, Proceedings*, volume 2556 of *Lecture Notes in Computer Science*. Springer, 2002.

[3] R. M. Amadio. On modelling mobility. *Theoretical Computer Science*, 240(1):147–176, 2000.

[4] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 147–162.

[5] L. Bettini, V. Bono, R. D. Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The klaim project: Theory and practice. In C. Priami, editor, *Global Computing: Programming Environments, Languages, Security and Analysis of Systems*, number 2874 in LNCS. Springer-Verlag, 2003.

[6] L. Bettini, R. De Nicola, G. Ferrari, and R. Pugliese. Interactive Mobile Agents in X-Klaim. In P. Ciancarini and R. Tolksdorf, editors, *Proc. of the 7th Int. IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 110–115, Stanford, 1998. IEEE Computer Society Press.

[7] L. Bettini, R. De Nicola, and R. Pugliese. Klava: a Java Package for Distributed and Mobile Applications. *Software — Practice and Experience*, 32:1365–1394, 2002.

[8] M. Boreale and R. De Nicola. Testing equivalences for mobile processes. *Journal of Information and Computation*, 120:279–303, 1995. Available as Report SI 92 RR 04, Università "La Sapienza" di Roma; an extended abstract appeared in *Proceedings of CONCUR '92*, LNCS 630.

[9] M. Boreale, R. De Nicola, and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 172:139–164, 2002.

[10] M. Boreale, R. D. Nicola, and R. Pugliese. Basic observables for processes. *Information and Computation*, 149(1):77–98, 1999.

[11] M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication interference in mobile boxed ambients. In Agrawal and Seth [2], pages 71–84.

[12] L. Cardelli. Abstractions for mobile computation. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS, pages 51–94. Springer, 1999.

[13] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. An extended abstract appeared in *Proceedings of FoSSaCS '98*, number 1378 of Lecture Notes in Computer Science, pages 140-155, Springer, 1998.

[14] G. Castagna and F. Z. Nardelli. The Seal Calculus Revisited: contextual equivalence and bisimilarity. In Agrawal and Seth [2], pages 85–96.

[15] I. Castellani. Process algebras with localities. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 945–1045. Elsevier Science, 2001.

[16] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In V. Arvind and R. Ramanujam, editors, *Proceedings of FSTTCS '98*, volume 1530 of *LNCS*, pages 90–101. Springer, Dec. 1998.

[17] S. Castellani, P. Ciancarini, and D. Rossi. The ShaPE of ShaDE: a coordination system. Technical Report UBLCS 96-5, Dip. di Scienze dell'Informazione, Univ. di Bologna, Italy, 1996.

[18] N. Davies, S. Wade, A. Friday, and G. Blair. L^2imbo: a tuple space based platform for adaptive mobile applications. In *Int. Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP'97)*, 1997.

[19] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[20] R. De Nicola, D. Gorla, and R. Pugliese. Bisimulations for a calculus for global computing. Draft, 2004.

[21] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of KLAIM-based calculi. In F. Corradini and J. Baeten, editors, *Proc. of EXPRESS'04*, ENTCS. Elsevier, 2004.

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

[23] D. Deugo. Choosing a Mobile Agent Messaging Model. In *Proc. of ISADS 2001*, pages 278–286. IEEE, 2001.

[24] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 406–421. Springer, 1996.

[25] C. Fournet and C. Laneve. Bisimulations in the join-calculus. *Theoretical Computer Science*, 266(1-2):569–603, 2001.

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

[27] J. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In L. Brim, P. Jancar, M. Kretínský, and A. Kucera, editors, *CONCUR*, volume 2421 of *LNCS*, pages 272–287. Springer, 2002.

[28] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. In *Proceedings of FoSSaCS '03*, volume 2620 of *LNCS*, pages 282–299. Springer, 2003. Full version as COGS Computer Science Technical Report, 2002:01.

[29] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995. An extract appeared in *Proceedings of FSTTCS '93*, LNCS 761.

[30] M. Merro and M. Hennessy. Bisimulation congruences in Safe Ambients. In *Proceedings of POPL '02*. ACM, 2002.

[31] M. Merro and F. Z. Nardelli. Bisimulation proof methods for mobile ambients. In *Proc. of ICALP'03*, LNCS. Springer, 2003. Full version as COGS Technical Report 2003:1, University of Sussex, Brighton.

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

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

[34] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[35] U. Montanari and M. Pistore. Finite state verification for the asynchronous pi-calculus. In R. Cleaveland, editor, *Proc. of TACAS'99*, volume 1579 of LNCS, pages 255–269. Springer, 1999.

[36] U. Nestmann and B. C. Pierce. Decoding choice encodings. *Journal of Information and Computation*, 163:1–59, 2000. Also available as report BRICS-RS-99-42, Universities of Aalborg and Århus, Denmark, 1999. An extended abstract appeared in the Proceedings of *CONCGUR '96*, LNCS 1119, pages 179–194.

[37] A. Park and P. Reichl. Personal Disconnected Operations with Mobile Agents. In *Proc. of 3rd Workshop on Personal Wireless Communications, PWC'98*, Tokyo, 1998.

[38] J. Parrow. An introduction to the pi-calculus. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.

[39] J. Riely and M. Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 266:693–735, 2001.

[40] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, LFCS, University of Edinburgh, 1993. CST-99-93 (also published as ECS-LFCS-93-266).

[41] P. Sewell, P. Wojciechowski, and B. Pierce. Location independence for mobile agents. In H. Bal, B. Belkhouche, and L. Cardelli, editors, *Proceedings of ICCL '98, Workshop on Internet Programming Languages (Chicago, IL, USA, May 13, 1998)*, volume 1686 of *LNCS*. Springer, Sept. 1999. Full version with title *Location-Independent Communication for Mobile Agents: a Two-Level Architecture* appeared as Technical Report 462, Computer Laboratory, University of Cambridge, April 1999.

[42] P. Thati, R. Ziaei, and G. Agha. A Theory of May Testing for Actors. In *Proc. of FMOODS'02*, pages 147–162. Kluwer, 2002.