Technical Report 07/2004, Dip. di Informatica, Univ. di Roma “La Sapienza”

Basic Observables for a Calculus for Global Computing*

Rocco De Nicola[1] Daniele Gorla[2] Rosario Pugliese[1]

[1]Dipartimento di Sistemi e Informatica, Università di Firenze

[2]Dipartimento di Informatica, Università di Roma “La Sapienza”

Abstract

We develop the semantic theory of a foundational language for modelling applications over global computers whose interconnection structure can be explicitly manipulated. Together with process distribution, process mobility and remote asynchronous communication through distributed data repositories, the language provides constructs for explicitly modelling inter-node connections and for dynamically activating and deactivating them. For the proposed language, we define natural notions of extensional observations and study their closure under operational reductions and/or language contexts to obtain *barbed congruence* and *may testing* equivalence. For such equivalences, we provide alternative characterizations in terms of a labelled *bisimulation* and a *trace* equivalence that can be used for actual proofs. We discuss how the language and its theory can be extended to include more sophisticated features that enable a finer control on the activation of connections. To asses practical usability of the semantic theory, we model a scenario for communications between mobile devices and use the introduced proof techniques to analyze it and verify some relevant properties.

*This paper is an extended and revised version of [15] and was mainly written when the second author was a PhD student in Florence.

Contents

1 Introduction 3

2 The Process Language tKlaim 4
2.1 Syntax 5
2.2 Operational Semantics 6

3 Observables, Closures and Equivalences 7

4 Bisimulation Equivalence 10
4.1 A Labelled Transition System for Labelled Bisimulation 10
4.2 Coincidence with Barbed Congruence 18

5 Trace Equivalence 20
5.1 A Labelled Transition System for Trace Equivalence 21
5.2 Soundness w.r.t. May Testing 24
5.3 Completeness w.r.t. May Testing 28

6 Controlling the Activation of a Connection 32
6.1 A Revised Labelled Transition System 33
6.2 A Bisimulation-based Characterisation of Barbed Congruence 36
6.3 On a Trace-based Characterisation of May Testing 45

7 An Example: Dynamic Connections in a Cellular Net 45

8 Conclusions and Related Work 50

1 Introduction

Programming global computational infrastructures for offering uniform services over wide area networks has become one of the main issues in Computer Science. Innovative theories, computational paradigms, linguistic mechanisms and implementation techniques have been proposed that have to face the challenges posed by issues like communication, cooperation, mobility, resource usage, security, failure handling, etc.. We have thus witnessed the birth of many calculi and kernel languages intended to support programming of global systems and to provide formal tools for reasoning over them. These formalisms in general provide constructs and mechanisms, at different abstraction levels, for modelling the execution contexts of the network where applications roam and run, for coordinating and monitoring the use of resources, for expressing process communication and mobility, and for specifying and enforcing security policies.

Much research effort has been devoted to study the impact of different communication and mobility paradigms, but little attention has been devoted to the modelling of the actual network underlying global computers as such. Usually, the model of the network implicitly originates from the linguistic choices concerning the mobility paradigm. All foundational languages proposed in the literature intend migration as the movement either of processes or of some other entities having executable content (such as nodes themselves). Thus, the former languages model the network as an evolving graph of fully connected nodes (see e.g., Dπ-calculus [23], KLAIM [13], Nomadic Pict [40], $\pi_{1\ell}$-calculus [1]) whereas the latter ones model the network as an evolving forest of trees (see e.g., Ambient [9] and its variants, DJoin [18], Homer [24], M-calculus [38], Seal [10]). In our view, both approaches do not convincingly model global computers (the Internet is neither a clique nor a forest of trees) and lack of flexibility (e.g. 'sharing of resources' is diffi cult to control and requires complex modelling).

In [16] we introduce a new modelling language that takes its origin from two formalisms with opposite objectives, namely the programming language X-KLAIM [3] and the π-calculus [29]. The former one is a full fledged programming language based on KLAIM [13], whereas the latter one is the generally recognized minimal common denominator of calculi for mobility. The resulting model has been called TKLAIM (*Topological* KLAIM); it retains the main features of KLAIM (distribution, remote operations, process mobility and asynchronous communication through distributed data spaces), but extends it with new constructs to flexibly model the interconnection structure underlying a network. TKLAIM provides two specifi c process primitives (inspired by those presented in [4]) to activate and deactivate inter-node connections. Connections become essential to perform remote operations: these are possible only if the node where they are initiated and the target one are directly connected.

Here, we develop the semantic theory of TKLAIM. We introduce two abstract semantics, *barbed congruence* and *may testing*, that are obtained as the closure under operational reductions and/or language contexts of the extensional equivalences induced by what we consider basic *observables* for global computers. For deciding the observables to use, we have been struggling with the following ones:

i. a specifi c node is up and running (i.e., it provides a datum of any kind)

ii. a specifi c information is available in (at least) a node,

iii. a specifi c information is present at a specifi c node.

Other calculi for global computers make use of (barbed) congruences induced by similar observables: for example, Ambient uses barbs that are somewhat related to *i.*; the barbs in Dπ-calculus instead, are strongly related to *iii.*. Within our framework, we prove that, by closing observations under any TKLAIM context, the three basic observables all yield the same congruence. This is already an indication of the robustness of the resulting semantic theories. Moreover, the observables are powerful enough to yield interesting semantic theories also when considering lower-level features, such as failures [16].

Of course, the step that comes next after defining equivalences as context closure is determining some alternative characterizations that would permit to better appreciate their discriminating power and to devise proof techniques that avoid universal quantification over contexts (that would render equivalence checking very hard).

In this paper, we focus on the barbed and may equivalences induced by the first basic observable (*a node is up and running*) and define alternative characterizations in terms of labelled *weak bisimilarity* and *trace* equivalence, resp.. Trace equivalence often suffices to express properties of interest for programs (e.g., those properties that can be expressed in terms of reachability of a particular state), whereas weak bisimilarity usually enjoys more effective proof techniques (and can be itself considered as a sound proof technique for the former). For these reasons we will study both equivalences and experiment with them on an example application.

To develop the alternative characterisations of the two abstract semantics of interest, we introduce and exploit a *labelled transition system* (LTS). A distinctive feature of our LTS is that labels indicate the resources a term offers or requires to the execution context for combined evolution, whereas usually they indicate the actions performed by the term. We define weak bisimilarity on top of this LTS and define trace equivalence on a slightly modified LTS. For both the equivalences we present soundness and completeness results with respect to barbed equivalence and may testing, respectively. The actual development of the alternative characterizations, although performed along the lines of similar results for CCS [30, 7] and π-calculus [36, 2], had to face problems raised by process distribution and mobility, by the explicit modelling of connections and by asynchrony.

We then focus on a smooth variant of TKLAIM where a handshake between the nodes involved is necessary to activate a connection. This mechanism is implemented by introducing a new process primitive that, by synchronizing with a connection request, authorises the activation of a new connection (this is similar to the so-called *co-capabilities* of Safe Ambients [27]). Instead, disconnections are still modeled as unilateral (and, then, asynchronous) events. Also in this finer scenario, we develop an alternative characterization of barbed equivalence in terms of weak bisimilarity and discuss on a trace-based proof-technique for may testing.

The rest of the paper is organized as follows. In Section 2 we present TKLAIM's syntax and reduction-based semantics. In Section 3 we define barbed congruence and may testing, while in Sections 4 and 5 we develop their alternative characterizations. In Section 6, we present an extension of the language and its theory to model a finer scenario where a handshaking between the involved nodes is needed to activate connections. In Section 7 we illustrate an example of the use of TKLAIM and of its semantic theories to state and prove properties of global computing applications. Finally, in Section 8 we conclude by also touching upon related work.

W.r.t. the extended abstract [15], here we give full details of proofs, we simplify the LTS (by reducing the number of labels), we present the variant where connections must be authorised and we give a sample application of both bisimulation and trace equivalence to a simplified real-life scenario.

2 The Process Language TKLAIM

In this section, we present the syntax of TKLAIM and its operational semantics based on a structural congruence and a reduction relation.

Nets:	N	$::=$	$\mathbf{0} \;\big|\; l :: C \;\big|\; \{l_1 \leftrightarrow l_2\} \;\big|\; N_1 \parallel N_2 \;\big|\; (\nu l)N$
Components:	C	$::=$	$\langle l \rangle \;\big|\; P \;\big|\; C_1 \mid C_2$
Processes:	P	$::=$	$\mathbf{nil} \;\big|\; a.P \;\big|\; P_1 \mid P_2 \;\big|\; *P$
Actions:	a	$::=$	$\mathbf{in}(!x)@u \;\big|\; \mathbf{in}(u_2)@u_1 \;\big|\; \mathbf{out}(u_2)@u_1 \;\big|\; \mathbf{eval}(P)@u$
			$\big|\; \mathbf{new}(l) \;\big|\; \mathbf{conn}(u) \;\big|\; \mathbf{disc}(u)$

Table 1: TKLAIM Syntax

2.1 Syntax

The syntax of TKLAIM is reported in Table 1, where we assume the existence of a countable set of *names*, ranged over by $l, l', \ldots, u, \ldots, x, y, \ldots$. Names provide the abstract counterpart of the set of *communicable* objects and can be used as localities or variables; notationally, we prefer letters $l, l', \ldots$ when we want to stress the use of a name as a locality, and $x, y, \ldots$ when we want to stress the use of a name as a variable. We will use u for variables and localities.

Nets, ranged over by $N, M, H, K, \ldots$, are finite collections of nodes and inter-node connections. 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 $\langle l \rangle$. *Connections* are pairs of node addresses $\{l_1 \leftrightarrow l_2\}$ stating that the nodes at address l_1 and l_2 are directly (and bidirectionally[1]) connected. In $(\nu l)N$, name l is private to N; the intended effect is that, if one considers the term $M \parallel (\nu l)N$, then locality l of N cannot be referred from within M. We consider restricted names as private network addresses and we shall arrange to work with nets where restricted names correspond to actual nodes. Therefore, in the sequel, we assume this condition holds for nodes created at the outset; for nodes created dynamically this follows from the operational semantics, see rule (R-NEW) in Table 4.

Processes, ranged over by $P, Q, R, \ldots$, are the TKLAIM active computational units and may be executed concurrently either at the same locality or at different localities. They are built from the inert process **nil** and from the basic actions by using prefixing, parallel composition and replication. *Actions* permit removing/adding data from/to node repositories (actions **in** and **out**), activating new threads of execution (action **eval**), creating new nodes (action **new**), and activating and deactivating connections (actions **conn** and **disc**). Notice that $\mathbf{in}(l)@l'$ differs from $\mathbf{in}(!x)@l'$ in that the former evolves only if datum $\langle l \rangle$ is present at l', whereas the latter accepts any datum. Indeed, $\mathbf{in}(l)@l'$ is a form of *name matching operator* reminiscent of LINDA's [21] pattern-matching.

Names occurring in TKLAIM processes and nets can be *bound*. More precisely, prefix $\mathbf{in}(!x)@u.P$ binds x in P; prefix $\mathbf{new}(l).P$ binds l in P, and, similarly, net restriction $(\nu l)N$ binds l in N. A name that is not bound is called *free*. The sets $fn(\cdot)$ and $bn(\cdot)$ of free and bound names of a term, respectively, are defined accordingly (their definitions are shown in Table 2). The set $n(\cdot)$ of names of a term is the union of its 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

[1]For the sake of simplicity, we assumed bidirectional connections; nevertheless, all the developed theory could be tailored to the framework where connections are directed.

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

C	$fn(\_)$	$bn(\_)$
$\langle l\rangle$	$\{l\}$	$\emptyset$
$C_1 \mid C_2$	$fn(C_1) \cup fn(C_2)$	$bn(C_1) \cup bn(C_2)$
$\mathbf{nil}$	$\emptyset$	$\emptyset$
$a.P$	$(fn(P) - bn(a)) \cup fn(a)$	$bn(P) \cup bn(a)$
$*P$	$fn(P)$	$bn(P)$

N	$fn(\_)$	$bn(\_)$
$\mathbf{0}$	$\emptyset$	$\emptyset$
$l :: C$	$\{l\} \cup fn(C)$	$bn(C)$
$\{l_1 \leftrightarrow l_2\}$	$\{l_1, l_2\}$	$\emptyset$
$N_1 \parallel N_2$	$fn(N_1) \cup fn(N_2)$	$bn(N_1) \cup bn(N_2)$
$(\nu l)N$	$fn(N) - \{l\}$	$bn(N) \cup \{l\}$

Table 2: Free and bound names

symbolic name A to the term W. We shall use notation $\widetilde{\cdot}$ to denote a possibly empty set of objects (e.g. $\widetilde{l}$ is a set of names). If $\widetilde{x} = \{x_1, \ldots, x_n\}$ and $\widetilde{y} = \{y_1, \ldots, y_m\}$, then $(\widetilde{x}, \widetilde{y})$ will denote the set of pairwise distinct elements $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$. 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 tailing occurrences of process **nil** and write $\prod_{j=1}^{n} W_j$ for the parallel composition of homologous terms (i.e., components or nets) W_j.

2.2 Operational Semantics

TKLAIM operational semantics relies on 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. The first eight laws are taken from the π-calculus (see, e.g., [35]). 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. Additionally, law (ABS) is the equivalent of law (PZERO) for '|' and law (CLONE) transforms a parallel between co-located components into a parallel between nodes; hence, monoidality of '|' can be obtained by relying on (PCOM) and (PASS). Laws (SELF), (BIDIR) and (CONNODE) are used to handle connections: the first one states that nodes are self-connected, the second one states that connections are bidirectional and the third one states that connections can be placed only between existing nodes.

The *reduction relation* is given in Table 4. In (R-OUT) and (R-EVAL), the existence of a connection between the nodes source and target of the action is necessary to place the spawned component. Notice that existence of the connection can only be checked at run-time: an approach like [23] does not fit well in a global computing setting because it relies on a typing mechanism that would require to statically know the whole net. (R-IN) and (R-MATCH) additionally require the existence of a matching datum in the target node. (R-MATCH) states that action $\mathbf{in}(l)@l_2$ consumes exactly the datum $\langle l\rangle$ at l_2, whereas (R-IN) states that action $\mathbf{in}(!\,x)@l_2$ can consume any $\langle l\rangle$ at l_2; l will then replace the free occurrences of x in the continuation of the process performing the action. (R-NEW) states that execution of action $\mathbf{new}(l')$ creates a new node at the restricted address l' and a connection with the creating node l. Finally, (R-CONN) and (R-DISC) deal with activation/deactivation of connections. In the first case, existence of the connected nodes is checked; in the second case,

(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 $l \notin fn(N_1)$
(GARB)	$(\nu l)\mathbf{0}$	$\equiv$	$\mathbf{0}$	
(REPL)	$l :: *P$	$\equiv$	$l :: P \mid *P$	
(ABS)	$l :: C$	$\equiv$	$l :: (C \vert \mathbf{nil})$	
(CLONE)	$l :: C_1 \vert C_2$	$\equiv$	$l :: C_1 \parallel l :: C_2$	
(SELF)	$l :: \mathbf{nil}$	$\equiv$	$l :: \mathbf{nil} \parallel \{l \leftrightarrow l\}$	
(BIDIR)	$\{l_1 \leftrightarrow l_2\}$	$\equiv$	$\{l_2 \leftrightarrow l_1\}$	
(CONNODE)	$\{l_1 \leftrightarrow l_2\}$	$\equiv$	$l_1 :: \mathbf{nil} \parallel \{l_1 \leftrightarrow l_2\}$	

Table 3: Structural Congruence

existence of the connection to be deactivated is checked.

TKLAIM adopts a LINDA-like [21] communication mechanism: communication is asynchronous and data are anonymous. Indeed, no synchronization takes place between (sending and receiving) processes, because their interactions are mediated by nodes, that act as data repositories. For the sake of simplicity, we only consider monadic data, but the semantic theories we develop could be smoothly extended to deal with tuples of data and with a full-blown LINDA-like *pattern matching* mechanism (these extensions, for example, will be exploited in Section 7).

If $N \longmapsto N'$, we shall say that N can perform a reduction step and that N' is a *reduct* of N. We shall use $\models\!\!\Rightarrow$ to denote the reflexive and transitive closure of $\longmapsto$.

3 Observables, Closures and Equivalences

In this section we present both a linear time and a branching time equivalence that yield sensible semantic theories for TKLAIM. The approach we follow relies on the definition of an *observable* (also called *barb*), namely a predicate that highlights the interaction capabilities of a term.

Definition 3.1 (Observables or barbs) *Predicate $N \downarrow l$ holds true if $N \equiv (\nu\widetilde{l})(N' \parallel l :: \langle l' \rangle)$, for some $\widetilde{l}$, N' and l' such that $l \notin \widetilde{l}$. Predicate $N \Downarrow l$ holds true if $N \models\!\!\Rightarrow N'$ for some N' such that $N' \downarrow l$.*

We have chosen the basic observables by taking inspiration from those used for the asynchronous π-calculus [2]. One may wonder if our choice is "correct" and argue that there are other alternative notions of basic observables that seem quite natural. We have already proposed a few alternative observables in the Introduction; later on, we shall prove that the congruences induced by such observables do coincide. This means that our results are quite independent from the observable chosen and vindicates our choice. We want to remark that, by using other kinds of observables, more equivalences could be captured. For example, in [7], some observables are introduced that capture *must testing* and *fair testing* in the context of CCS.

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

(R-EVAL)
$$l_1 :: \mathbf{eval}(P_2)@l_2.P_1 \,\|\, \{l_1 \leftrightarrow l_2\} \;\longmapsto\; l_1 :: P_1 \,\|\, \{l_1 \leftrightarrow l_2\} \,\|\, l_2 :: P_2$$

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

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

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

(R-CONN)
$$l_1 :: \mathbf{conn}(l_2).P \,\|\, l_2 :: \mathbf{nil} \;\longmapsto\; l_1 :: P \,\|\, \{l_1 \leftrightarrow l_2\}$$

(R-DISC)
$$l_1 :: \mathbf{disc}(l_2).P \,\|\, \{l_1 \leftrightarrow l_2\} \;\longmapsto\; l_1 :: P \,\|\, l_2 :: \mathbf{nil}$$

(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: TKLAIM Operational Semantics

We use observables to define equivalence relations that identify those nets that cannot be taken apart by any basic observation in any execution context.

Definition 3.2 (Contexts) *A* context $C[\cdot]$ *is a* TKLAIM *net with an occurrence of a hole* $[\cdot]$ *to be filled in with any net. Formally,*

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

By relying on laws (EXT) and (RCOM), every context can be put in a particular syntactic form, with all the restrictions at top-level; thus, when convenient, we shall use $(\nu\widetilde{l})([\cdot] \,\|\, K)$ – for any net K without restriction – to identify all those contexts $C[\cdot]$ that are structurally equivalent to it.

Definition 3.3 *A binary relation* $\Re$ *between nets is*

- barb preserving, *if* $N \Re M$ *and* $N \Downarrow l$ *imply* $M \Downarrow l$*;*
- reduction closed, *if* $N \Re M$ *and* $N \longmapsto N'$ *imply* $M \models\!\!\Rightarrow M'$ *and* $N' \Re M'$*, for some* M'*;*
- context closed, *if* $N \Re M$ *implies* $C[N] \Re C[M]$*, 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 would be too weak: indeed, the set of barbs of a net may change during computations or when interacting with the external environment. Moreover, for the sake of compositionality, our touchstone equivalences should also be congruences. These requirements lead us to the following definitions.

Definition 3.4 (May testing) $\simeq$ *is the largest symmetric, barb preserving and context closed relation between nets.*

Definition 3.5 (Barbed congruence) $\cong$ *is the largest symmetric, barb preserving, reduction and context closed relation between nets.*

The above definition of barbed congruence is the standard one, see [25, 35]. May testing is, instead, usually defined in terms of *observers*, *computations* and *success of a computation* [17]. Intuitively, two nets are may testing equivalent if they cannot be distinguished by any external observer taking note of the data offered by the observed nets. Here we prove that such an alternative characterization can be given for $\simeq$.

Definition 3.6 (Observers) Observers, *ranged over by O, O', O_1, ..., are nets whose processes and nodes can use the distinct and reserved locality name* `test` *as address of a node or as target of operations.*

Definition 3.7 (Computations) Computations *from $N \parallel O$ are (possibly infinite) sequences of reductions of the form $N \parallel O \, (\equiv (\nu\widetilde{l_0})(N_0 \parallel O_0)) \longmapsto (\nu\widetilde{l_1})(N_1 \parallel O_1) \longmapsto \cdots$. Such a computation is* successful *if there is some $i \geq 0$ such that $O_i \equiv O' \parallel \mathtt{test} :: \langle\rangle$ and $\mathtt{test} \notin \widetilde{l_i}$. We write N $\textsc{may}$ O whenever there exists a successful computation from $N \parallel O$.*

Definition 3.8 *$N \simeq' M$ if, for every observer O, it holds that N $\textsc{may}$ O if and only if M $\textsc{may}$ O.*

Proposition 3.9 $\cong \; \subset \; \simeq \; = \; \simeq'$.

Proof: By definition, it trivially follows that $\cong$ is contained in $\simeq$. The inclusion is strict: a pair of may-testing equivalent nets that are not barbed congruent is

- $(\nu l)(l :: \mathbf{out}(\,)@l'.\mathbf{out}(\,)@l \parallel l_1 :: \mathbf{in}(\,)@l.\mathbf{out}(\,)@l_1 \parallel l_2 :: \mathbf{in}(\,)@l.\mathbf{out}(\,)@l_2)$
- $(\nu l)(l :: \mathbf{out}(\,)@l \parallel l_1 :: \mathbf{in}(\,)@l.\mathbf{out}(\,)@l'.\mathbf{out}(\,)@l_1 \parallel l_2 :: \mathbf{in}(\,)@l.\mathbf{out}(\,)@l'.\mathbf{out}(\,)@l_2)$.

Such nets mimic the CCS processes $(\overline{l'}.\overline{l} \mid l.\overline{l_1} \mid l.\overline{l_2})\backslash_l$ and $(\overline{l} \mid l.\overline{l'}.\overline{l_1} \mid l.\overline{l'}.\overline{l_2})\backslash_l$ that, written in terms of the internal choice operator '$\oplus$', become $\overline{l'}.(\overline{l_1} \oplus \overline{l_2})$ and $\overline{l'}.\overline{l_1} \oplus \overline{l'}.\overline{l_2}$. It is well-known [17] that such processes are may testing equivalent but not barbed congruent.

We now prove that $\simeq = \simeq'$. We start with $\simeq \subseteq \simeq'$. Let $N \simeq M$ and take an observer O such that N $\textsc{may}$ O. Then, by context closure, $N \parallel O \simeq M \parallel O$ and, by barb preservation, $N \parallel O \Downarrow \mathtt{test}$ (that comes from N $\textsc{may}$ O) implies that $M \parallel O \Downarrow \mathtt{test}$. Since `test` is a name occuring only in O (by definition of observers), it must be M $\textsc{may}$ O, as required.

Vice versa, to prove that $\simeq' \subseteq \simeq$, it suffices to prove that $\simeq'$ is barb preserving and context closed. Let $N \simeq' M$. For barb preservation, let $N \Downarrow l$ and consider $O \triangleq \mathtt{test} :: \mathbf{in}(!x)@l.\mathbf{out}()@\mathtt{test} \parallel \{\mathtt{test} \leftrightarrow l\}$. Then, N $\textsc{may}$ O that, by hypothesis, implies M $\textsc{may}$ O. Now, because of freshness of `test`, this is possible only if $M \Downarrow l$. For context closure, the proof is by induction on the structure of the context $\mathcal{C}[\cdot]$. The base case is trivial. For the inductive case, we have two possibilities:

- $\mathcal{C}[\cdot] \triangleq \mathcal{D}[\cdot] \parallel H$. By induction, we may assume that $\mathcal{D}[N] \simeq' \mathcal{D}[M]$. Let O be an observer such that $\mathcal{C}[N]$ $\textsc{may}$ O. We now consider the observer $H \parallel O$; by Definition 3.8, by induction and by the fact that $\mathcal{D}[N]$ $\textsc{may}$ $H \parallel O$, we have that $\mathcal{D}[M]$ $\textsc{may}$ $H \parallel O$. By rule (PASS) and because $\equiv \subseteq \simeq'$, this implies $\mathcal{C}[M]$ $\textsc{may}$ O.

- $\mathcal{C}[\cdot] \triangleq (\nu l)\mathcal{D}[\cdot]$. Since l is bound, we can assume, up-to alpha-equivalence, that $l \notin n(O)$ for any observer O. Now, $\mathcal{C}[N]$ $\textsc{may}$ O if and only if $\mathcal{D}[N]$ $\textsc{may}$ O (and similarly when replacing N with M). By induction, $\mathcal{D}[N] \simeq' \mathcal{D}[M]$; this suffices to conclude. ∎

The problem with the definitions of barbed congruence and may testing is that context closure makes it hard to prove equivalences due to the universal quantification over contexts. In the following sections, we shall provide two alternative characterisations of $\cong$ and $\simeq$, as a *bisimulation-based* and as a *trace-based* equivalence, respectively.

Before doing this, we show that we can change the basic observables without changing the congruences they induce; this proves the robustness of our touchstone equivalences and supports our choice. Recalling from the Introduction, other two reasonable observables in our framework are existence of a specific (visible) datum at some node of a net and existence of a specific datum at a specific node of a net.

Definition 3.10 (Alternative Touchstone Equivalences) *Let $\cong_1$, $\cong_2$ $\simeq_1$ and $\simeq_2$ be the barbed congruences and the may testing equivalences obtained by replacing the observable of Definition 3.1, respectively, with the following ones:*

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

We now prove that, thanks to context closure, $\cong_1$ and $\cong_2$ coincide with $\cong$, and that $\simeq_1$ and $\simeq_2$ coincide with $\simeq$.

Proposition 3.11 $\cong_1 = \cong_2 = \cong$ *and* $\simeq_1 = \simeq_2 = \simeq$.

Proof: Notice that we only need to consider barb preservation. Indeed, context and reduction closure (the latter one only in the case of barbed congruences) are ensured by definition. We explicitly present the case for barbed congruences; the proofs for may testing can then be rephrased straightforwardly.

$\cong_2 \subseteq \cong_1$. 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 $N \Downarrow l$, i.e. $N \models\!\!\Rightarrow (\nu\widetilde{l})(N' \parallel l :: \langle l' \rangle)$. Then $M \Downarrow l$, otherwise the context $[\cdot] \parallel l'' :: \mathbf{in}(!x)@l.\mathbf{out}(l'')@l'' \parallel \{l \leftrightarrow l''\}$, for l'' fresh, would break $\cong_1$.

$\cong \subseteq \cong_2$. Let $N \cong M$ and $N \Downarrow_l \langle l' \rangle$, i.e. $N \models\!\!\Rightarrow (\nu\widetilde{l})(N' \parallel l :: \langle l' \rangle)$. Then $M \Downarrow_l \langle l' \rangle$, otherwise the context $[\cdot] \parallel l'' :: \mathbf{in}(l')@l.\mathbf{out}(l'')@l'' \parallel \{l \leftrightarrow l''\}$, for l'' fresh, would break $\cong$. ∎

4 Bisimulation Equivalence

We now provide a more tractable characterisation of barbed congruence by means of a *labelled bisimulation*. To this aim, we start by presenting an alternative (but equivalent w.r.t. the reductions of Table 4) semantics for TKLAIM by means of a labelled transition system. We then present the bisimulation-based characterisation of barbed congruence and prove that the two equivalences do coincide.

4.1 A Labelled Transition System for Labelled Bisimulation

The labelled transition system (LTS) makes apparent the possible contributions that a net can offer/require in a computation. The *labelled transition relation*, $\xrightarrow{\alpha}$, is defined as the least relation over nets induced by the inference rules in Table 5. Labels take the form

$$\beta \;::=\; l_1 \frown l_2 \;\big|\; \langle l \rangle @ l_1 : l_2 \qquad\qquad \alpha \;::=\; \tau \;\big|\; \beta \;\big|\; \exists ?\beta \;\big|\; (\nu l)\, \langle l \rangle @ l_1 : l_2$$

(LTS-LINK)
$$\{l_1 \leftrightarrow l_2\} \xrightarrow{l_1 \curvearrowright l_2} l_1 :: \mathbf{nil} \parallel l_2 :: \mathbf{nil}$$

(LTS-DATUM)
$$l_1 :: \langle l \rangle \xrightarrow{\langle l \rangle @ l_1 : l_1} l_1 :: \mathbf{nil}$$

(LTS-CONN)
$$l_1 :: \mathbf{conn}(l_2).P \xrightarrow{\exists ? l_2 \curvearrowright l_2} l_1 :: P \parallel \{l_1 \leftrightarrow l_2\}$$

(LTS-DISC)
$$l_1 :: \mathbf{disc}(l_2).P \xrightarrow{\exists ? l_1 \curvearrowright l_2} l_1 :: P \parallel l_2 :: \mathbf{nil}$$

(LTS-EVAL)
$$l_1 :: \mathbf{eval}(P_2)@l_2.P_1 \xrightarrow{\exists ? l_1 \curvearrowright l_2} l_1 :: P_1 \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: P_2$$

(LTS-OUT)
$$l_1 :: \mathbf{out}(l)@l_2.P \xrightarrow{\exists ? l_1 \curvearrowright l_2} l_1 :: P \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l \rangle$$

(LTS-IN)
$$l_1 :: \mathbf{in}(!\, x)@l_2.P \xrightarrow{\exists ?\ \langle l \rangle @ l_2 : l_1} l_1 :: P[{}^{l}\!/\!{}_{x}] \parallel \{l_1 \leftrightarrow l_2\}$$

(LTS-MATCH)
$$l_1 :: \mathbf{in}(l)@l_2.P \xrightarrow{\exists ?\ \langle l \rangle @ l_2 : l_1} l_1 :: P \parallel \{l_1 \leftrightarrow l_2\}$$

(LTS-NEW)
$$l :: \mathbf{new}(l').P \xrightarrow{\tau} (\nu l')(l :: P \parallel \{l \leftrightarrow l'\})$$

(LTS-OFFER)
$$\frac{N_1 \xrightarrow{\langle l \rangle @ l_2 : l_2} N_1' \qquad N_2 \xrightarrow{l_1 \curvearrowright l_2} N_2'}{N_1 \parallel N_2 \xrightarrow{\langle l \rangle @ l_2 : l_1} N_1' \parallel N_2'}$$

(LTS-OPEN)
$$\frac{N \xrightarrow{\langle l \rangle @ l_2 : l_1} N' \qquad l \notin \{l_1, l_2\}}{(\nu l)N \xrightarrow{(\nu l)\ \langle l \rangle @ l_2 : l_1} N'}$$

(LTS-COMPL)
$$\frac{N_1 \xrightarrow{\exists ? \beta} N_1' \quad N_2 \xrightarrow{\beta} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

(LTS-RES)
$$\frac{N \xrightarrow{\alpha} N' \qquad l \notin n(\alpha)}{(\nu l)N \xrightarrow{\alpha} (\nu l)N'}$$

(LTS-PAR)
$$\frac{N_1 \xrightarrow{\alpha} N_2 \qquad bn(\alpha) \cap fn(N) = \emptyset}{N_1 \parallel N \xrightarrow{\alpha} N_2 \parallel N}$$

(LTS-STRUCT)
$$\frac{N \equiv N_1 \quad N_1 \xrightarrow{\alpha} N_2 \quad N_2 \equiv N'}{N \xrightarrow{\alpha} N'}$$

Table 5: A Labelled Transition System

In the sequel, we shall write $(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2$ to denote label $\langle l \rangle @ l_1 : l_2$, if $\widetilde{l} = \emptyset$, and label $(\nu l)\ \langle l \rangle @ l_1 : l_2$ otherwise (i.e. if $\widetilde{l} = \{l\}$). Moreover, we let $bn(\alpha)$ be $\widetilde{l}$ if $\alpha = (\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2$ and be $\emptyset$ otherwise; $fn(\alpha)$ and $n(\alpha)$ are defined accordingly.

Let us now explain the intuition behind the labels of the LTS and some key rules. Label α in $N \xrightarrow{\alpha} N'$ can be

τ **:** this means that N may perform a reduction step to become N' (see Proposition 4.4).

$l_1 \curvearrowright l_2$ **:** this means that a direct connection between nodes l_1 and l_2 is offered (see (LTS-LINK)).

$(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2$ **:** this means that a datum $\langle l \rangle$ located at l_1 is offered to a process located at l_2 (see (LTS-DATUM) and (LTS-OFFER)). Moreover, according to whether $\widetilde{l} = \{l\}$ or $\widetilde{l} = \emptyset$, l is restricted in N or not (see (LTS-OPEN)).

$\exists ? l_1 \curvearrowright l_2$**:** this means that there is a process located at l_1 that needs existence of a connection with l_2 (see rules (LTS-DISC), (LTS-OUT) and (LTS-EVAL)). Moreover, if $l_1 = l_2$, it can also be that a process in the net requires the existence of node l_2 (see rule (LTS-CONN) and the structural rule (SELF)). In both cases, such requirement is satisfied by a 'complementary' label $l_1 \curvearrowright l_2$ (see rule (LTS-COMPL)).

$\exists ?\,\langle l\rangle @ l_2 : l_1$: this means that there is a process located at l_1 that needs to retrieve the datum $\langle l\rangle$ from l_2 (see (LTS-IN) and (LTS-MATCH)). For the retrieval to succeed, a direct connection between such nodes is also needed (see (LTS-COMPL)).

To sum up briefly, labels of kind $l_1 \curvearrowright l_2$ and $(\nu\widetilde{l})\,\langle l\rangle @ l_1 : l_2$ point out the structure of a net, and account for the resources (connections and data) the net can 'offer' to the execution context as a contribution to combined evolution. On the other hand, labels of kind $\exists ?\beta$ describe the resources a net 'requires' to the execution context for combined evolution. For example, (LTS-OUT) should be read as: "process $\mathbf{out}(l)@l_2.P$ running at l_1 is willing to send a component at l_2; when such intention is concretised (i.e., when the execution context provides the connection needed), l_1 will host process P for execution and will run in a net where the connection $\{l_1 \leftrightarrow l_2\}$ does exist and the datum $\langle l\rangle$ is placed at l_2". Indeed, since label $\exists ?l_1 \curvearrowright l_2$ requires the existence of the connection $\{l_1 \leftrightarrow l_2\}$, every execution context satisfying this requirement allows the sending net to assume the existence of the connection required; this enables the net to place the datum at the target node. Rules (LTS-EVAL), (LTS-IN), (LTS-MATCH), (LTS-CONN) and (LTS-DISC) should be interpreted similarly.

Notably, pointing out what the context should provide for an action to be performed, rather than the action itself, permits using the same label for all those actions with similar requirements (viz. **out**, **eval** and **disc**), instead of having a different label for each possible action. This has a positive impact on all those proofs that proceed by case analysis on the labels of the LTS.

Rule (LTS-OPEN) signals extrusion of bound names; as in some presentation of the π-calculus (see, e.g., [35]), this rule is used to investigate the capability of processes to export bound names, rather than to actually extend the scope of bound names. This is instead achieved through the structural law (EXT); in fact, in (LTS-COMPL) labels do not carry any restriction on names, whose scope must have been previously extended. (LTS-RES), (LTS-PAR) and (LTS-STRUCT) are standard.

Notice that the LTS of Table 5 may appear unnecessarily complicated as a tool to define the operational semantics of TKLAIM: consider, e.g., the right hand side of the axioms. Nevertheless, it is adequate as a tool to establish the alternative, more tractable, characterisation of barbed bisimulation (the only transitions having a counterpart in the reduction semantics are those labelled by τ). Indeed, the complications in the operational rules of Table 5 resemble those arisen in [39] when defining an 'equivalent' LTS depending on the reduction semantics of a calculus. However, in [39] only simple calculi are considered and it would be challenging to investigate if the approach can be satisfactory extended to TKLAIM.

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 say that N can perform a α-step. Moreover, we shall usually denote relation composition by juxtaposition; thus, e.g., $N \xrightarrow{\alpha}\xrightarrow{\alpha'} M$ means that there exists a net N' such that $N \xrightarrow{\alpha} N' \xrightarrow{\alpha'} M$. Given a relation $\mathcal{R}$, notation $\not\mathcal{R}$ means that $\mathcal{R}$ does not hold (e.g. $N \not\xrightarrow{\alpha} N'$ means that N cannot reduce to N' by performing α). As usual, we let $\Rightarrow$ stand for $\xrightarrow{\tau}^{*}$ and $\overset{\alpha}{\Longrightarrow}$ to stand for $\Rightarrow\xrightarrow{\alpha}\Rightarrow$; finally, $\overset{\hat{\alpha}}{\Longrightarrow}$ denotes $\Rightarrow$, if $\alpha = \tau$, and $\overset{\alpha}{\Longrightarrow}$, otherwise.

We conclude the presentation of the LTS by highlighting some of its properties. First, we connect transitions with the syntactical form of the net performing them. Then, we characterise all the possible combined executions of a net N within a context $(\nu\widetilde{l})([\cdot] \parallel K)$ in terms of the evolutions of the net and of the context separately. Finally, we show that the LTS is 'correct' w.r.t. the operational semantics of TKLAIM based on $\longmapsto$.

Proposition 4.2 *The following facts hold:*

1. *if* $N \xrightarrow{l_1 \curvearrowright l_2} N'$, *then* $N \equiv N' \parallel \{l_1 \leftrightarrow l_2\}$;
2. *if* $N \xrightarrow{\langle l\rangle @ l_1 : l_2} N'$, *then* $N \equiv N' \parallel l_1 :: \langle l\rangle \parallel \{l_1 \leftrightarrow l_2\}$;
3. *if* $N \xrightarrow{(\nu l)\,\langle l\rangle @ l_1 : l_2} N'$, *then* $N \equiv (\nu l)(N' \parallel l_1 :: \langle l\rangle \parallel \{l_1 \leftrightarrow l_2\})$ *and* $l \notin \{l_1, l_2\}$.
4. *if* $N \xrightarrow{\exists ? l_1 \curvearrowright l_2} N'$, *one of the following possibilities must hold:*
   (a) *if* $l_1 = l_2$, *then* $N \equiv (\nu\widetilde{l})(N'' \parallel l_2 :: \mathbf{conn}(l_2).P)$, *for* $l_2 \notin \widetilde{l}$, *and* $N' \equiv (\nu\widetilde{l})(N'' \parallel l_2 :: P)$;
   (b) $N \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: \mathbf{disc}(l_2).P)$, *for* $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$, *and* $N' \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: P \parallel l_2 :: \mathbf{nil})$;
   (c) $N \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: \mathbf{eval}(P_2)@l_2.P_1)$, *for* $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$, *and* $N' \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: P_1 \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: P_2)$;
   (d) $N \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: \mathbf{out}(l)@l_2.P)$, *for* $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$, *and* $N' \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: P \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle)$.
5. *if* $N \xrightarrow{\exists ?\,\langle l\rangle @ l_2 : l_1} N'$, *one of the following possibilities must hold:*
   (a) $N \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: \mathbf{in}(!x)@l_2.P)$, *for* $\{l, l_1, l_2\} \cap \widetilde{l} = \emptyset$, *and* $N' \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: P[^l/_x] \parallel \{l_1 \leftrightarrow l_2\})$;
   (b) $N \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: \mathbf{in}(l)@l_2.P)$, *for* $\{l, l_1, l_2\} \cap \widetilde{l} = \emptyset$, *and* $N' \equiv (\nu\widetilde{l})(N'' \parallel l_1 :: P \parallel \{l_1 \leftrightarrow l_2\})$.

Proof: By definition of the LTS and a straightforward induction on the depth of the shortest inference for the judgement in the hypothesis. ∎

Proposition 4.3 $(\nu\widetilde{l})(N \parallel K) \xrightarrow{\alpha} \overline{N}$ *if and only if one of the following cases holds:*

1. $(\nu\widetilde{l})N \xrightarrow{\alpha} (\nu\widetilde{l'})N'$ *and* $\overline{N} \equiv (\nu\widetilde{l'})(N' \parallel K)$
2. $N \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_1 : l_1} N'$, $K \xrightarrow{l_1 \curvearrowright l_2} K'$ *and* $\overline{N} \equiv (\nu\widetilde{l''})(N' \parallel K')$, *where* $\alpha = (\nu l)\,\langle l\rangle @ l_1 : l_2$ *and* $\widetilde{l''} = \widetilde{l} - \{l\}$, *if* $\widetilde{l'} = \emptyset$ *and* $l \in \widetilde{l}$, *while* $\alpha = (\nu\widetilde{l'})\,\langle l\rangle @ l_1 : l_2$ *and* $\widetilde{l''} = \widetilde{l}$, *otherwise*
3. $N \xrightarrow{\exists ? l_1 \curvearrowright l_2} N'$, $K \xrightarrow{l_1 \curvearrowright l_2} K'$, $\overline{N} \equiv (\nu\widetilde{l})(N' \parallel K')$ *and* $\alpha = \tau$
4. $N \xrightarrow{\exists ?\,\langle l\rangle @ l_2 : l_1} N'$, $K \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_2 : l_1} K'$, $\widetilde{l'} \cap fn(N) = \emptyset$, $\overline{N} \equiv (\nu\widetilde{l}, \widetilde{l'})(N' \parallel K')$ *and* $\alpha = \tau$
5. $N \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\,\langle l\rangle @ l_2 : l_1} N'$, $K \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_2 : l_2} K'$, $\widetilde{l'} \cap fn(N) = \emptyset$, $\overline{N} \equiv (\nu\widetilde{l}, \widetilde{l'})(N' \parallel K')$ *and* $\alpha = \tau$
6. $N \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\,\langle l\rangle @ l_2 : l_1} N'$, $K \xrightarrow{l_2 \curvearrowright l_1} K'$, $\widetilde{l'} \cap fn(K) = \emptyset$, $\overline{N} \equiv (\nu\widetilde{l}, \widetilde{l'})(N' \parallel K')$ *and* $\alpha = \tau$
7. *one of the previous cases with K in place of N and vice versa.*

Proof: The "if" part is trivial, by using the LTS of Table 5. We explicitly consider only three significant cases.

2. If $\widetilde{l'} = \emptyset$, then by applying rule (LTS-OFFER) we get that $N \parallel K \xrightarrow{\langle l\rangle @ l_1 : l_2} N' \parallel K'$. Now, if $l \in \widetilde{l}$, then the thesis follows by applying rule (LTS-OPEN); otherwise, the thesis follows by applying rule (LTS-RES). If $\widetilde{l'} \neq \emptyset$, i.e. $\widetilde{l'} = \{l\}$, then from the hypothesis $N \xrightarrow{(\nu \widetilde{l'})\ \langle l\rangle @ l_1 : l_1} N'$; by Proposition 4.2(3), we get that there exists a net N_1 such that $N \equiv (\nu l)N_1$ and $N_1 \xrightarrow{\langle l\rangle @ l_1 : l_1} N'$. By applying the structural rule (EXT), we get $N \parallel K \equiv (\nu l)(N_1 \parallel K)$. Now, by applying rule (LTS-OFFER) we get that $N_1 \parallel K \xrightarrow{\langle l\rangle @ l_1 : l_2} N' \parallel K'$. Then, the thesis easily follows by applying rules (LTS-OPEN) and (LTS-STRUCT).

4. By $K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1} K'$ and by Proposition 4.2(3), we get that there exists a net K'' such that $K \equiv (\nu l)K''$ and $K'' \xrightarrow{\langle l\rangle @ l_1 : l_1} K'$. By applying the structural rule (EXT), we get $N \parallel K \equiv (\nu l)(N \parallel K'')$. Now, by applying rule (LTS-COMPL) we get that $N \parallel K'' \xrightarrow{\tau} N' \parallel K'$ and the thesis easily follows by applying rules (LTS-RES) and (LTS-STRUCT).

7. By structural congruence, we have that $(\nu\widetilde{l})(N \parallel K) \equiv (\nu\widetilde{l})(K \parallel N)$; then we fall in one of the fi rst six cases.

The “only if” part would have been easily proved by induction on the shortest inference of $\xrightarrow{\alpha}$, if rule (LTS-STRUCT) was not present. To properly handle such a rule, we consider a slightly different (but still equivalent) LTS where rule (LTS-STRUCT) is restricted in such a way that $N \equiv N_1$ can only be derived by using just a single axiom (or its symmetric version) from Table 3. Through the rest of the proof, (LTS-STRUCT) will refer to this revised rule. Notice that transitivity of $\equiv$ may require repeated applications of (LTS-STRUCT), whereas closure under language contexts can be mimicked by properly interleaving the application of (LTS-STRUCT), (LTS-RES) and (LTS-PAR).

Now, we can reason by induction on the depth of the shortest inference for $\xrightarrow{\alpha}$. We have three base cases (of depth 2); in all of them, $\widetilde{l} = \emptyset$ and the hypotheses are axioms from Table 5. We analyse the last rule used in the inference:

- (LTS-PAR): we fall in case 1. of this Lemma.
- (LTS-OFFER): we fall in case 2. of this Lemma.
- (LTS-COMPL): we fall in case 3. or 4. of this Lemma.

For the inductive step, we reason by case analysis on the last rule applied in the inference. The cases for (LTS-PAR), (LTS-OFFER) and (LTS-COMPL) are easily adapted from the base case (no inductive hypothesis is needed).

(LTS-RES): let $\widetilde{l} = (l, \widetilde{l'})$; then, $(\nu\widetilde{l'})(N \parallel K) \xrightarrow{\alpha} \overline{N}'$ and $\overline{N} \triangleq (\nu l)\overline{N}'$, for $l \notin n(\alpha)$. By induction on $(\nu\widetilde{l'})(N \parallel K) \xrightarrow{\alpha} \overline{N}'$, that has a shorter inference, we fall in one of the cases of this Lemma. In the same case falls also the inference for $(\nu\widetilde{l})(N \parallel K) \xrightarrow{\alpha} \overline{N}$.

(LTS-OPEN): the situation is similar to the previous one, but $\overline{N} \triangleq \overline{N}'$ and $l \in fn(\alpha')$; so, $\alpha \neq \tau$ and hence we can only fall in cases 1 or 2 (or their symmetric versions). In the same case falls the inference from $(\nu\widetilde{l})(N \parallel K)$, with $\alpha = (\nu l)\alpha'$.

(LTS-STRUCT): we reason by case analysis on the axiom of Table 3 used by the rule. If reflexivity of $\equiv$ or axiom (ALPHA) is used, we rely on a trivial induction; otherwise, we have the following possibilities:

(PZERO): $\widetilde{l} = \emptyset$ and we fall in case 1. of this Lemma.

(PCOM): $\widetilde{l} = \emptyset$ and $K \parallel N \xrightarrow{\alpha} \overline{N}'$, for $\overline{N}' \equiv \overline{N}$. By induction on $K \parallel N \xrightarrow{\alpha} \overline{N}'$, that has a shorter inference, we fall in one of the cases of this Lemma. Now, if the induction yields one of the first six cases, the original net $N \parallel K$ evolves according to the symmetric case and vice versa.

(PASS): again, $\widetilde{l} = \emptyset$; moreover, $N \triangleq N_1 \parallel N_2$ and $N_1 \parallel (N_2 \parallel K) \xrightarrow{\alpha} \overline{N}'$, for $\overline{N}' \equiv \overline{N}$. We now apply induction and reason on the case in which the latter transition falls:

1. $N_1 \xrightarrow{\alpha} N_1'$ and $\overline{N}' \triangleq N_1' \parallel (N_2 \parallel K)$: we still easily fall in case 1.
2. $N_1 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_1 : l_1} N_1'$, $N_2 \parallel K \xrightarrow{l_1 \curvearrowright l_2} \overline{N}''$ and $\overline{N}' \triangleq N_1' \parallel \overline{N}''$: by induction, $N_2 \parallel K \xrightarrow{l_1 \curvearrowright l_2}$ is only possible when either $N_2 \xrightarrow{l_1 \curvearrowright l_2}$ or $K \xrightarrow{l_1 \curvearrowright l_2}$; then, $(N_1 \parallel N_2) \parallel K$ evolves according to cases 1. or 2., respectively.

3., 5., 6.: similar to the previous case.

4. $N_1 \xrightarrow{\exists ?\ \langle l\rangle @ l_1 : l_2} N_1'$, $N_2 \parallel K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1} \overline{N}''$ and $\overline{N}' \triangleq (\nu l)(N_1' \parallel \overline{N}'')$: by induction, $N_2 \parallel K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1}$ can be inferred in four ways:
   - $N_2 \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to case 1.
   - $K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to case 4.
   - $N_2 \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_2}$ and $K \xrightarrow{l_2 \curvearrowright l_1}$: then, $N \parallel K$ evolves according to case 6.
   - $N_2 \xrightarrow{l_2 \curvearrowright l_1}$ and $K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_2}$: then, $N \parallel K$ evolves according to case 5.

7. The symmetric of cases 2. and 3. are dealt with like cases 2. and 3. themselves; the symmetric of case 4. proceeds like case 2. The remaining cases are as follows:
   1. $N_2 \parallel K \xrightarrow{\alpha} \overline{N}''$ and $\overline{N}' \triangleq N_1 \parallel \overline{N}''$: we apply induction to $N_2 \parallel K \xrightarrow{\alpha} \overline{N}''$; the case for $(N_1 \parallel N_2) \parallel K$ is the same as that obtained in this latter inductive step.
   5. $N_1 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} N_1'$, $N_2 \parallel K \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \overline{N}''$ and $\overline{N}' \triangleq (\nu\widetilde{l'})(N_1' \parallel \overline{N}'')$: by induction, $N_2 \parallel K \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ can be inferred in four ways:
      - $N_2 \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to case 1.
      - $K \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 5.
      - $N_2 \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ and $K \xrightarrow{l_2 \curvearrowright l_1}$: then, $N \parallel K$ evolves according to case 6.
      - $N_2 \xrightarrow{l_2 \curvearrowright l_1}$ and $K \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 4.
   6. $N_1 \xrightarrow{l_2 \curvearrowright l_1} N_1'$, $N_2 \parallel K \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \overline{N}''$ and $\overline{N}' \triangleq (\nu\widetilde{l'})(N_1' \parallel \overline{N}'')$: by induction, $N_2 \parallel K \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ can be inferred in four ways:
      - $N_2 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to case 1.
      - $K \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 6.
      - $N_2 \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ and $K \xrightarrow{(\nu\widetilde{l'})\langle l\rangle @ l_2 : l_2}$: then, $N \parallel K$ evolves according to case 5.
      - $N_2 \xrightarrow{(\nu\widetilde{l'})\langle l\rangle @ l_2 : l_2}$ and $K \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 4.

symmetric version of (PASS): similar to the previous one, but now $K \triangleq K_1 \parallel K_2$ and $(N \parallel K_1) \parallel K_2 \xrightarrow{\alpha} \overline{N}'$. We only illustrate here the more elaborated cases:

4. $N \parallel K_1 \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1} \overline{N}''$, $K_2 \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \overline{K}_2'$ and $\overline{N}' \triangleq (\nu l)(\overline{N}'' \parallel K_2')$: by induction, $N \parallel K_1 \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1}$ can be inferred in four ways:
   - $N \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 4.
   - $K_1 \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 1.
   - $N \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_2}$ and $K_1 \xrightarrow{l_2 \curvearrowright l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 5.
   - $N \xrightarrow{l_2 \curvearrowright l_1}$ and $K_1 \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_2}$: then, $N \parallel K$ evolves according to the symmetric of case 6.
5. $N \parallel K_1 \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \overline{N}''$, $K_2 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} K_2'$ and $\overline{N}' \triangleq (\nu\widetilde{l'})(\overline{N}'' \parallel K_2')$: by induction, $N \parallel K_1 \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ can be inferred in four ways:
   - $N \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to case 5.
   - $K_1 \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 1.
   - $N \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ and $K_1 \xrightarrow{l_2 \curvearrowright l_1}$: then, $N \parallel K$ evolves according to case 4.
   - $N \xrightarrow{l_2 \curvearrowright l_1}$ and $K_1 \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 6.
6. $N \parallel K_1 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \overline{N}''$, $K_2 \xrightarrow{l_2 \curvearrowright l_1} K_2'$ and $\overline{N}' \triangleq (\nu\widetilde{l'})(\overline{N}'' \parallel K_2')$: by induction, $N \parallel K_1 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ can be inferred in four ways:
   - $N \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to case 6.
   - $K_2 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 1.
   - $N \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2}$ and $K_1 \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$: then, $N \parallel K$ evolves according to the symmetric of case 5.
   - $N \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1}$ and $K_1 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2}$: then, $N \parallel K$ evolves according to case 4.

(RCOM): $\widetilde{l} = (l_1, l_2, \widetilde{l'})$ and $(\nu l_2)(\nu l_1)(\nu\widetilde{l'})(N \parallel K) \xrightarrow{\alpha} \overline{N}'$, for $\overline{N}' \equiv \overline{N}$; then, by induction, we can conclude that $(\nu l_1)(\nu l_2)(\nu\widetilde{l'})(N \parallel K)$ evolves correspondingly.

(EXT): $\widetilde{l} = \emptyset$; moreover, $K \triangleq (\nu l)\overline{K}$ and $(\nu l)(N \parallel \overline{K}) \xrightarrow{\alpha} \overline{N}'$, for $\overline{N}' \equiv \overline{N}$ and $l \notin fn(N)$. Now, we can apply induction to $(\nu l)(N \parallel \overline{K}) \xrightarrow{\alpha} \overline{N}'$ and conclude that $N \parallel K$ evolves in the same way as $(\nu l)(N \parallel \overline{K})$; just notice that, whenever $\overline{K} \xrightarrow{\langle l\rangle @ l_2 : l_1} K'$ arises upon induction, we obtain $K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1} K'$.

symmetric version of (EXT): similar to the previous one. Notice that now $\widetilde{l} = \{l\}$; moreover, every time $(\nu l)K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1} K'$ arises, it will be replaced by $K \xrightarrow{\langle l\rangle @ l_2 : l_1} K'$.

symmetric versions of (REPL), (CLONE), (SELF) or (CONNODE): we can build a no longer inference for $N \parallel K \xrightarrow{\alpha} \overline{N}$ where the symmetric versions of (REPL)/(CLONE)/(SELF)/(CONNODE) are not used at all. Thus, we can easily conclude by relying on one of the previous cases. ∎

Proposition 4.4 *$N \longmapsto M$ if and only if $N \xrightarrow{\tau} M$.*

Proof: Both directions are proved by an easy induction on the shortest inference of the judgements. The ‘only if’ part is simple: it has a base case for every axiom of Table 4 and, for the inductive step, we can exploit the similarity between (R-PAR) and (LTS-PAR), between (R-RES) and (LTS-RES), and between (R-STRUCT) and (LTS-STRUCT).

The ‘if’ part only considers judgements for τ-labelled transitions: for the base case, we need to consider rules (LTS-NEW) and (LTS-COMPL). The first case is simple, because of the similarity between (LTS-NEW) and (R-NEW). For the second case, we have that $N \triangleq N_1 \parallel N_2$ where $N_1 \xrightarrow{\exists ? \beta} N_1'$, $N_2 \xrightarrow{\beta} N_2'$ and $M \triangleq N_1' \parallel N_2'$. If β is of the form $\exists ? l_1 \curvearrowright l_2$, we conclude by exploiting cases (1) and (4) of Proposition 4.2; If β is of the form $\exists ?\ \langle l \rangle @ l_2 : l_1$, we conclude by exploiting cases (2) and (5) of the same Proposition. The inductive case is simple, since it only relies on rules (LTS-PAR), (LTS-RES) and (LTS-STRUCT). ∎

We can now introduce the alternative characterization of $\cong$ in terms of a labelled *bisimilarity*. To this aim, we first define the ‘minimal’ net enabling the evolution of a net performing a label of kind $\exists ? \beta$; formally,

$$\textsc{Net}(\beta) \triangleq \begin{cases} \{l_1 \leftrightarrow l_2\} & \text{if } \beta = l_1 \curvearrowright l_2 \\ \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l \rangle & \text{if } \beta = \langle l \rangle @ l_2 : l_1 \end{cases}$$

Definition 4.5 (Bisimilarity) *A symmetric relation $\mathfrak{R}$ between* TKLAIM *nets is a* (weak) bisimulation *if, for each $N \ \mathfrak{R}\ M$ and $N \xrightarrow{\alpha} N'$, it holds that:*

1. *if $\alpha \in \{\tau, l_1 \curvearrowright l_2, (\widetilde{\nu l})\ \langle l \rangle @ l_1 : l_1\}$, then $M \overset{\hat{\alpha}}{\Longrightarrow} M'$ and $N' \ \mathfrak{R}\ M'$, for some M';*
2. *if $\alpha = \exists ? \beta$, then $M \parallel \textsc{Net}(\beta) \Rightarrow M'$ and $N' \ \mathfrak{R}\ M'$, for some M'.*

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

Bisimilarity requires that labels of kind $l_1 \curvearrowright l_2$ and $(\widetilde{\nu l})\ \langle l \rangle @ l_1 : l_1$ must be replied to with the same label (possibly with some additional τ-step). This is necessary since such labels describe the structure of the net (its data and connections); to be equivalent, two nets must have at least the same structure. In doing this, notice that labels of kind $(\widetilde{\nu l})\ \langle l \rangle @ l_1 : l_2$ for $l_1 \neq l_2$ can be ignored, since they result from the combination of labels $\{l_1 \leftrightarrow l_2\}$ and $(\widetilde{\nu l})\ \langle l \rangle @ l_1 : l_1$, see rule (LTS-OFFER).

Labels of kind $\exists ? \beta$ are ‘requirements’ to the execution context; thus, they are handled differently. For example, requiring the existence of a connection (e.g., to send a component) expressed by $N \xrightarrow{\exists ? l_1 \curvearrowright l_2} N'$ can be simulated by a net M in a context where l_1 and l_2 are connected through the execution of some τ-steps that lead to some M' equivalent to N'. Indeed, since we want our bisimulation to be a congruence, a context that provides a connection between the source and the target nodes of the sending action must not tell N and M apart.

Notably, though in TKLAIM processes can occur as arguments in process actions (**eval**), the LTS and the bisimulation we developed do not use labels containing processes. Thus, the bisimulation relies only on a standard quantification over names (when considering labels of kind $\exists ?\ \langle l \rangle @ l_2 : l_1$) and we strongly conjecture that it is decidable, under proper assumptions: techniques similar to those in [31] could be used here. The presence of rule (LTS-STRUCT) in the LTS does not compromise the tractability of $\approx$; obviously, (LTS-STRUCT) can be dropped, if one is prepared to have more rules in the LTS.

4.2 Coincidence with Barbed Congruence

The first 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 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 4.5 is replaced by the (compound) relation $\equiv \mathfrak{R} \equiv$. Lemma 4.6 shows that a bisimulation up-to $\equiv$ can be used as a sound proof-technique for labelled bisimulation.

Lemma 4.6 *Let $\mathfrak{R}$ be a bisimulation up-to $\equiv$; then, $\mathfrak{R} \subseteq \approx$.*

Proof: We first prove that $\equiv \mathfrak{R} \equiv$ is a bisimulation. Let $N \equiv \mathfrak{R} \equiv M$; this means that there exist N_1 and M_1 such that $N \equiv N_1 \mathfrak{R} M_1 \equiv M$. Take $N \xrightarrow{\alpha} N'$; since $N \equiv N_1$, by (LTS-STRUCT), we also have that $N_1 \xrightarrow{\alpha} N'$. Now, we reason by case analysis on α. If $\alpha \in \{\tau, l_1 \curvearrowright l_2, (\nu\widetilde{l})\,\langle l\rangle\,@\,l_1 : l_1\}$, then we must show that there exists M' such that $M \overset{\hat{\alpha}}{\Longrightarrow} M'$ and $N' \equiv \mathfrak{R} \equiv M'$. Since, by hypothesis, $\mathfrak{R}$ is a bisimulation up-to $\equiv$, we have that $M_1 \overset{\hat{\alpha}}{\Longrightarrow} M'$ and $N' \equiv \mathfrak{R} \equiv M'$ for some M'. Since $M_1 \equiv M$, by (LTS-STRUCT), we have that $M \overset{\hat{\alpha}}{\Longrightarrow} M'$ and the thesis is proved. If $\alpha = \exists?\beta$, we must show that there exists M' such that $M \parallel \mathrm{NET}(\beta) \Rightarrow M'$ and $N' \equiv \mathfrak{R} \equiv M'$. Since, by hypothesis, $\mathfrak{R}$ is a bisimulation up-to $\equiv$, the fact that $N_1 \xrightarrow{\exists?\beta} N'$ and $N_1 \mathfrak{R} M_1$ implies that $M_1 \parallel \mathrm{NET}(\beta) \Rightarrow M'$ and $N' \equiv \mathfrak{R} \equiv M'$. Since $\equiv$ is a congruence, the thesis follows from the fact that $M_1 \equiv M$ and by (LTS-STRUCT). The symmetric case (for $M \xrightarrow{\alpha} M'$) is similar.

Now, if $N \mathfrak{R} M$, by reflexivity of $\equiv$, we have that $N \equiv \mathfrak{R} \equiv M$; since $\equiv \mathfrak{R} \equiv$ is a bisimulation, we have that $N \approx M$. ∎

Lemma 4.7 $\approx$ *is context closed.*

Proof: By the previous lemma, it suffices to prove that

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

is a bisimulation up-to $\equiv$. First, notice that $C[N] \equiv (\nu\widetilde{l})(N \parallel K)$, for K restriction free and a suitable $(\nu\widetilde{l})$. Then, by (LTS-STRUCT), any transition $C[N] \xrightarrow{\alpha} \bar{N}$ corresponds to a transition $(\nu\widetilde{l})(N \parallel K) \xrightarrow{\alpha} \bar{N}$. We must show that this transition can be matched by a transition from $C[M] \equiv (\nu\widetilde{l})(M \parallel K)$. According to Proposition 4.3 we have to examine twelve cases. The details are omitted, as they can be inferred from the proof of (the similar, but more complicated) Lemma 6.5. ∎

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

Proof: By Lemma 4.7, we know that $\approx$ is context closed. Thus, we only need to prove that $\approx$ is barb preserving and reduction closed. To prove that $\approx$ is barb preserving, let $N \Downarrow l$; by Definition 3.1 and construction of the LTS, this means that $N \xRightarrow{(\nu\widetilde{l})\,\langle l'\rangle\,@\,l:l}$, for some l' and $\widetilde{l}$. By hypothesis, we get that $M \xRightarrow{(\nu\widetilde{l})\,\langle l'\rangle\,@\,l:l}$; thus, by Proposition 4.2(1), $M \Downarrow l$. To prove that $\approx$ is reduction closed, let $N \longmapsto N'$; by Proposition 4.4, this implies that $N \xrightarrow{\tau} N'$. By hypothesis, we can find a M such that $N' \approx M'$ and $M \Rightarrow M'$; again by Proposition 4.4, $M \models\!\!\Rightarrow M'$. ∎

We now want to prove the converse, namely that all barbed congruent nets are also bisimilar. To this aim, we need some preliminary technicalities. First, we introduce the notation

$$\text{GO } l \text{ DO } a \text{ THEN } P$$

to denote a process that migrates at l to perform action a and then comes back to its starting location to execute P. Formally, GO l DO a THEN P running at l' is a shortcut for

$$\mathbf{conn}(l).\mathbf{eval}(a.\mathbf{eval}(\mathbf{disc}(l).P)@l')@l$$

Then, we define the standard *internal choice* operator, to non-deterministically select for execution exactly one between two processes, as follows:

$$P \oplus Q \;\triangleq\; \mathbf{new}(l).\mathbf{out}()@l.(\ \mathbf{in}()@l.P \mid \mathbf{in}()@l.Q\ )$$

It is easy to prove that $l' :: P \oplus Q$ can only reduce to either $l' :: P \parallel K$ or $l' :: Q \parallel K$ (or one of their reducts), where $K \approx \mathbf{0}$.

These kinds of processes are used to prove the following key Lemma. It states that we can throw away a fresh locality hosting a restricted datum from two barbed congruent nets and the resulting nets are bisimilar.

Lemma 4.9 *Let* $(\nu l)(N \parallel l_f :: \langle l \rangle) \;\cong\; (\nu l)(M \parallel l_f :: \langle l \rangle)$ *and* l_f *be fresh for* N, M *and* l*; then,* $N \approx M$.

Proof: By Lemma 4.6, it suffices to prove that

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

is a bisimulation up-to $\equiv$. We omit the details of the proof because it proceeds as the (more complicated) proof of Lemma 6.7. ∎

Theorem 4.10 (Completeness of $\approx$ w.r.t. $\cong$) *If* $N \cong M$ *then* $N \approx M$.

Proof: By Lemma 4.6, it suffices to prove that $\cong \cup \approx$ is a bisimulation up-to $\equiv$. Take $N \cong M$ and a transition $N \xrightarrow{\alpha} N'$; we then reason by case analysis on α.

$\alpha = \tau$. The thesis follows from reduction closure.

$\alpha = \langle l \rangle @ l_1 : l_1$. We consider the context

$$C[\cdot] \triangleq (\nu l')([\cdot] \parallel \{l_f \leftrightarrow l_1\} \parallel l_f :: \mathbf{in}(l)@l_1.\mathbf{disc}(l_1).(\mathbf{out}(l')@l_f \oplus \mathbf{nil}))$$

for l_f fresh, and the reduction $C[N] \models\!\Rightarrow \mathcal{D}[N'] \triangleq \overline{N}$, where

$$\mathcal{D}[\cdot] \triangleq (\nu l')([\cdot] \parallel l_f :: \mathbf{out}(l')@l_f \oplus \mathbf{nil})$$

By context and reduction closure, $C[M] \models\!\Rightarrow \overline{M}$ and $\overline{N} \;\cong\; \overline{M}$. This fact implies that $M \xRightarrow{\langle l \rangle @ l_1 : l_1} M'$, for some M', otherwise $\overline{M}$ would not be able to exhibit a barb at l_f (whereas $\overline{N}$ can). Now consider the reduction $\overline{N} \models\!\Rightarrow \mathcal{D}'[N'] \triangleq \overline{N}'$, with

$$\mathcal{D}'[\cdot] \;\triangleq\; (\nu l')([\cdot] \parallel l_f : \langle l' \rangle) \;\parallel\; (\nu l'')(\{l_f \leftrightarrow l''\} \parallel l_f :: \mathbf{in}()@l''.\mathbf{nil}),$$

which is obtained by exploiting the definition of $\oplus$ and resolving the choice in favour of the left hand side (l'' is the locality created to implement '$\oplus$'). By reduction closure, it must be that $\overline{M} \models\!\Rightarrow \overline{M}'$ and $\overline{N}' \cong \overline{M}'$; because of freshness of l_f, this implies that $\overline{M}' \equiv \mathcal{D}'[M'']$, for some M'' such that $M' \Rightarrow M''$. Now, it is easy to prove that $\mathcal{D}'[N'] \approx (\nu l')(N' \parallel l_f : \langle l' \rangle)$ (and similarly for M''). Moreover, by using Theorem 4.8, we can replace $\approx$ with $\cong$; thus, $(\nu l')(N' \parallel l_f : \langle l' \rangle) \;\cong\; \mathcal{D}'[N'] \triangleq \overline{N}' \;\cong\; \overline{M}' \equiv \mathcal{D}'[M''] \;\cong\; (\nu l')(M'' \parallel l_f : \langle l' \rangle)$. Since $\equiv \subseteq \cong$ and $\cong$ is transitive, by Lemma 4.9, we get $N' \approx M''$; this suffices to conclude.

$\alpha = (\nu l)\, \langle l \rangle @ l_1 : l_1$ **.** We consider the context

$$C[\cdot] \triangleq [\cdot] \parallel \{l_f \leftrightarrow l_1\} \parallel l_f :: \mathbf{in}(!x)@l_1.\mathbf{disc}(l_1).(\mathbf{out}(x)@l_f \oplus \mathbf{nil})$$

for l_f fresh, and proceed as in the previous case. Notice that M must eventually exhibit a restricted datum at l_1. Indeed, the presence of any datum at l_1 is ascertained by action $\mathbf{in}(!x)@l_1$. Moreover, at least one restricted datum must be present at l_1, otherwise l_f would exhibit only free data in any evolution of $C[M]$, whereas l_f exhibits a restricted datum in the chosen evolution of $C[N]$.[2] Such a restricted datum can then be alpha-converted to l to obtain that $M \overset{\alpha}{\Longrightarrow} M''$ and $N' \approx M''$, for some M''.

$\alpha = l_1 \curvearrowright l_2$**.** Consider the context

$$C[\cdot] \triangleq (\nu l)([\cdot] \parallel l_f :: \textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2)\ \textsc{Then}\ (\mathbf{out}(l)@l_f \oplus \mathbf{nil}))$$

and proceed like before.

$\alpha = \exists ?\beta$**.** We consider the context $C[\cdot] \triangleq [\cdot] \parallel \textsc{Net}(\beta)$ and the reduction $C[N] \longmapsto N'$. Then, by context and reduction closure, $C[M] \models\!\Rightarrow M'$ and $N' \cong M'$, for some M'. This suffices to conclude (see Definition 4.5(2)). ∎

From Theorems 4.8 and 4.10 we get the wanted result.

Corollary 4.11 (Alternative Characterization of Barbed Congruence) $\approx\, = \,\cong$.

5 Trace Equivalence

In this section, we develop an alternative characterization of may testing. For some well-known process calculi, may testing coincides with trace equivalence [17, 5, 6]; in this section, we show how a similar result is obtained also in the setting of tKLAIM. To this aim, we first need to slightly tune the LTS of Section 4.1 to better deal with may testing; we then present the trace-based characterisation of $\simeq$ and prove that the resulting equivalence is sound and complete w.r.t. $\simeq$.

To carry out proofs in a simpler way, in all this section we assume that observers cannot perform remote input actions. This does not reduce the observational power of an observer, since a remote input, say at l_1 from l_2, can be implemented by first migrating to l_2, by performing there a local input and by finally coming back to l_2. Formally, every observer O is translated in an observer

[2] To see that there must be an evolution of $C[M]$ producing a restricted datum at l_1 (and, therefore, at l_f), consider the following context

$$[\cdot] \parallel \{l_f \leftrightarrow l'_f\} \parallel \prod_{l' \in fn(M)} l'_f :: \mathbf{in}(l')@l_f.\mathbf{out}()@l'_f$$

where l'_f is a fresh locality. The choosen evolution of $C[N]$ will never enable the production of a datum at l'_f, since we assumed that bound names are different from the free ones; thus, M cannot only produce free data at l_1, otherwise it would not be equivalent to N.

(LTS-CONN)

$$l_1 :: \mathbf{conn}(l_2).P \xrightarrow{\exists ? l_2} l_1 :: P \parallel \{l_1 \leftrightarrow l_2\}$$

(LTS-EST)

$$\frac{N_1 \xrightarrow{\exists ? l} N_1' \quad N_2 \xrightarrow{l \curvearrowright l} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

(LTS-BIN)

$$\frac{N \xrightarrow{\exists ?\ \langle l \rangle @ l_2 : l_1} N' \qquad l \notin fn(N)}{N \xrightarrow{\exists ?(\nu l)\ \langle l \rangle @ l_2 : l_1} N' \parallel l :: \mathbf{nil}}$$

(LTS-DATUMREQ)

$$\frac{N_1 \xrightarrow{l_1 \curvearrowright l_2} N_1' \qquad N_2 \xrightarrow{\exists ?(\nu \widetilde{l})\ \langle l \rangle @ l_2 : l_1} N_2'}{N_1 \parallel N_2 \xrightarrow{\exists ?(\nu \widetilde{l})\ \langle l \rangle @ l_2 : l_2} N_1' \parallel N_2'}$$

(LTS-CONNREQ)

$$\frac{N_1 \xrightarrow{(\nu \widetilde{l})\ \langle l \rangle @ l_1 : l_1} N_1' \qquad N_2 \xrightarrow{\exists ?(\nu \widetilde{l})\ \langle l \rangle @ l_1 : l_2} N_2'}{N_1 \parallel N_2 \xrightarrow{\exists ? l_1 \curvearrowright l_2} (\nu \widetilde{l})(N_1' \parallel N_2')}$$

plus all rules from Table 5, but (LTS-CONN); moreover, rule (LTS-COMPL) now also includes the side condition $bn(\beta) = \emptyset$

Table 6: An Enhanced LTS for Trace Equivalence

without remote inputs, $\langle\!| O |\!\rangle$, where $\langle\!| \cdot |\!\rangle$ acts homomorphically over all net constructors, except for $l :: C$ that is translated to $l :: \langle\!| C |\!\rangle_l$ as follows:

$$\begin{array}{lcl@{\qquad}lcl}
\langle\!| \langle u \rangle |\!\rangle_l & \triangleq & \langle u \rangle & \langle\!| C_1 | C_2 |\!\rangle_l & \triangleq & \langle\!| C_1 |\!\rangle_l \mid \langle\!| C_2 |\!\rangle_l \\
\langle\!| \mathbf{nil} |\!\rangle_l & \triangleq & \mathbf{nil} & \langle\!| *P |\!\rangle_l & \triangleq & * \langle\!| P |\!\rangle_l
\end{array}$$

$$\langle\!| a.P |\!\rangle_l \triangleq \begin{cases} \text{GO } l' \text{ DO } a \text{ THEN } \langle\!| P |\!\rangle_l & \text{if } a = \mathbf{in}(\cdot)@l' \\ \mathbf{eval}(\langle\!| Q |\!\rangle_{l'})@l'.\langle\!| P |\!\rangle_l & \text{if } a = \mathbf{eval}(Q)@l' \\ a.\langle\!| P |\!\rangle_l & \text{otherwise} \end{cases}$$

The observers $\langle\!| O |\!\rangle$ and O behave "in the same way", in the sense that they can observe the same net behaviours, as stated by the following Proposition. Clearly, this implies that, if we adapt Definition 3.8 by only considering this class of observers, we still obtain $\simeq'$ that, by Proposition 3.9, coincides with $\simeq$. Therefore, without loss of accuracy, we shall use these three definitions of may-testing interchangeably.

Proposition 5.1 *N MAY O if and only if N MAY $\langle\!| O |\!\rangle$.*

Proof: We first notice that $\langle\!| \cdot |\!\rangle$ is easily derived from the encoding of cKLAIM in LCKLAIM presented in [14], where we prove that such an encoding enjoys semantical equivalence w.r.t. $\cong$. By straightforwardingly adapting such a proof, we can prove that $O \cong \langle\!| O |\!\rangle$; thus, by context closure, $N \parallel O \cong N \parallel \langle\!| O |\!\rangle$ that, by barb preservation, implies the claim of this Proposition. ▮

5.1 A Labelled Transition System for Trace Equivalence

To properly handle may testing, we modify the LTS of Section 4.1, as reported in Table 6. First, we introduce *bound input* labels of the form $\exists ?(\nu l)\ \langle l \rangle @ l_1 : l_2$ to take note that the received name

l is bound. Second, it is now convenient to let action **conn**(l) yield a new label $\exists ?l$ (see the new version of rule (LTS-CONN)), that still synchronises with label $l \curvearrowright l$ (see rule (LTS-EST)). Thus, the syntax of labels can now be written as follows:

$$\beta \ ::= \ l_1 \curvearrowright l_2 \ \big| \ (\nu\widetilde{l})\,\langle l\rangle @ l_1 : l_2 \qquad\qquad \alpha \ ::= \ \tau \ \big| \ \beta \ \big| \ \exists ?\beta \ \big| \ \exists ?l$$

Of course, $bn(\alpha)$ is extended by letting $bn(\exists ?(\nu l)\,\langle l\rangle @ l_1 : l_2) = \{l\}$, whereas $fn(\alpha)$ and $n(\alpha)$ are extended accordingly. In what follows, we shall write $\exists ?^{-}$ to refer labels of kind $\exists ?\beta$ and $\exists ?l$.

Intuitively, rule (LTS-BIN) checks whether the name l received via an input does not occur free in the receiving net. In that case, it can be (additionally) assumed that l has been extruded by a sending net; hence, it is the address of a node and a bound input label is generated (similarly to the π-calculus [5, 6]). However, to avoid that $N_1 \xrightarrow{(\nu l)\,\langle l\rangle @ l_1 : l_2} N_1'$ and $N_2 \xrightarrow{\exists ?(\nu l)\,\langle l\rangle @ l_1 : l_2} N_2'$ synchronise by means of the previous formulation of (LTS-COMPL), that would lead to $N_1' \parallel N_2'$ instead of the expected $(\nu l)(N_1' \parallel N_2')$, a new side condition to rule (LTS-COMPL) is needed to still force structural scope extension of bound names.

Rule (LTS-DATUMREQ) states that, if node l_1 requires the existence of a (possibly restricted datum) $\langle l\rangle$ at l_2 and there exists a connection between l_1 and l_2 in the net, then it suffi ces to require to the execution context to provide datum $\langle l\rangle$ at l_2. Similarly, rule (LTS-CONNREQ) states that, if node l_2 requires the existence of a (possibly restricted datum) $\langle l\rangle$ at l_1 and such a datum is already in the net, then it suffi ces to require to the execution context to provide the connection between l_1 and l_2. To better understand the latter two rules, compare them with Proposition 4.3(5) and (6).

For the modifi ed LTS, it is easy to prove that Propositions 4.4 and 4.2 still hold. Instead, Proposition 4.3 has to be tuned as follows; the proof of the following result is very similar to the proof of Proposition 4.3 and, thus, it is omitted.

Proposition 5.2 $(\nu\widetilde{l})(N \parallel K) \xrightarrow{\alpha} \overline{N}$ *if and only if one of the following cases holds:*

1., 2., 3., 6.: like the corresponding cases of Proposition 4.3

4. $N \xrightarrow{\exists ?(\nu\widetilde{l''})\,\langle l\rangle @ l_2 : l_1} N',\ K \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_2 : l_1} K',\ \widetilde{l'} \cap fn(N) = \emptyset,\ \widetilde{l''} \subseteq \widetilde{l'},\ \overline{N} \equiv (\nu\widetilde{l},\widetilde{l'})(N' \parallel K')$ *and* $\alpha = \tau$

5. $N \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists ?(\nu\widetilde{l''})\,\langle l\rangle @ l_2 : l_1} N',\ K \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_2 : l_2} K',\ \widetilde{l'} \cap fn(N) = \emptyset,\ \widetilde{l''} \subseteq \widetilde{l'},\ \overline{N} \equiv (\nu\widetilde{l},\widetilde{l'})(N' \parallel K')$ *and* $\alpha = \tau$

7. $N \xrightarrow{l_1 \curvearrowright l_2} N', K \xrightarrow{\exists ?(\nu\widetilde{l'})\,\langle l\rangle @ l_2 : l_1} K', \widetilde{l'} \cap fn(N) = \emptyset, \overline{N} \equiv (\nu\widetilde{l})(N' \parallel K')$ *and* $\alpha = \exists ?(\nu\widetilde{l'})\langle l\rangle @ l_2 : l_2$

8. $N \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_1 : l_1} N',\ K \xrightarrow{\exists ?(\nu\widetilde{l''})\,\langle l\rangle @ l_2 : l_1} K',\ \widetilde{l'} \cap fn(K) = \emptyset,\ \widetilde{l''} \subseteq \widetilde{l'},\ \overline{N} \equiv (\nu\widetilde{l},\widetilde{l'})(N' \parallel K')$ *and* $\alpha = \exists ?l_2 \curvearrowright l_1$

9. $N \xrightarrow{\exists ?l} N', K \xrightarrow{l \curvearrowright l} K', \overline{N} \equiv (\nu\widetilde{l})(N' \parallel K')$ *and* $\alpha = \tau$

10. *one of the previous cases with K in place of N and vice versa.*

Notation 5.3 As a matter of notation, we shall use ϕ to range over visible labels (i.e. labels different from τ) and σ to range over (possibly empty) sequences of visible labels. As usual, $N \overset{\epsilon}{\Longrightarrow}$ denotes $N \Rightarrow$ and $N \overset{\phi\cdot\sigma}{\Longrightarrow}$ denotes $N \overset{\phi}{\Longrightarrow} \overset{\sigma}{\Longrightarrow}$.

Traditionally, trace equivalence relates N and M 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 vice versa. In an asynchronous setting [6, 11], this requirement must be properly weakened, since the discriminating power of asynchronous contexts is weaker: for example, the traditional formulation of trace equivalence would distinguish in TKLAIM nets $l :: \mathbf{in}(!x)@l_1.\mathbf{in}(!y)@l_2$ and $l :: \mathbf{in}(!y)@l_2.\mathbf{in}(!x)@l_1$, which are indeed may testing equivalent. Like in [6], a weaker trace-based equivalence can be defined by relying on a pre-order $\preceq$ on traces (rather than using identity).

Definition 5.4 (Trace Equivalence) $\asymp$ *is the largest symmetric relation between* TKLAIM *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 suitable pre-order $\preceq$ such that may testing is exactly captured by $\asymp$. The intuition behind $\sigma' \preceq \sigma$ (sometimes written as $\sigma \succeq \sigma'$) is that, if a context can interact with a net that exhibits σ, then the context can interact with any net that exhibits σ' as well, see Proposition 5.7. For TKLAIM, $\preceq$ is obtained as the reflexive and transitive closure of the relation $\preceq_0$ defined in Table 7. The first four laws have been inspired by [6], while the last five ones are strictly related to inter-node connections. The relation $\preceq_0$ 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 $\exists?(\nu l)\,\langle l\rangle @ l_1 : l_2$. In this case, the information that l is a fresh received name must be kept in the remaining trace. The formal definition is

$$\begin{array}{rcll} (\nu\widetilde{l})\sigma & \triangleq & \sigma & \text{if } \widetilde{l} \cap fn(\sigma) = \emptyset \\ (\nu l)(\phi \cdot \sigma) & \triangleq & \begin{cases} \exists?(\nu l)\,\langle l\rangle @ l_1 : l_2 \,\cdot \sigma & \\ \phi \cdot (\nu l)\sigma & \\ UNDEF & \end{cases} & \begin{array}{l} \text{if } \phi = \exists?\,\langle l\rangle @ l_2 : l_1 \text{ and } l \notin \{l_1, l_2\} \\ \text{if } l \notin n(\phi) \text{ and } (\nu l)\sigma \neq UNDEF \\ \text{otherwise} \end{array} \end{array}$$

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 $\exists?(\nu l)\,\langle l\rangle @ l_2 : l_1 \cdot \langle l\rangle @ l_3 : l_4$ performed by a N, the input action cannot be erased. Indeed, since l is fresh (see rule (LTS-BIN)), N cannot get knowledge of l without performing the input and, consequently, cannot perform the action $\langle l\rangle @ l_3 : l_4$. On the other hand, if N can receive l via an additional communication between another pair of nodes, say l_5 and l_6 (thus, it can perform action $\exists?\,\langle l\rangle @ l_6 : l_5$ just after $\exists?(\nu l)\,\langle l\rangle @ l_2 : l_1$), then the first input action can be erased and $\exists?(\nu l)\,\langle l\rangle @ l_6 : l_5 \cdot \langle l\rangle @ l_3 : l_4 \preceq_0 \exists?(\nu l)\,\langle l\rangle @ l_2 : l_1 \cdot \exists?\,\langle l\rangle @ l_6 : l_5 \cdot \langle l\rangle @ l_3 : l_4$.

We now briefly comment on the laws in Table 7. (L1) states that labels representing 'requirements' cannot be directly observed. (L2) states that the execution of a 'requirement' action can be delayed along computations without being noticed by any observer. (L3) states that two adjacent 'complementary' actions can be deleted. (L4) states that, like in the π-calculus [5, 6], may testing is unable to distinguish free addresses from bound ones. (L5) is used to guarantee existence of a node at address l in a net resulting from the execution of an action ϕ, as described by function $\Upsilon(\cdot)$; so, actions requiring the connection $\{l \leftrightarrow l\}$ are always enabled in such a net. (L6) states that, if a net satisfies the requirement $\exists?l \curvearrowright l$ after an action β, then it can also satisfy the requirement before β, since the node at l was already present; clearly, this is possible only if l has not been introduced by β, as described by function $\Upsilon(\cdot)$. Law (L7) states that a label of kind $(\nu\widetilde{l})\,\langle l\rangle @ l_1 : l_2$, for $l_1 \neq l_2$, is in practice a shortcut for label $l_1 \curvearrowright l_2$ followed by label $(\nu\widetilde{l})\,\langle l\rangle @ l_1 : l_1$. Finally, laws (L8) and (L9) state that, if a process located at l_1 can retrieve a datum $\langle l\rangle$ locally and then migrate at l_2, then a process located at l_2 can retrieve $\langle l\rangle$ remotely, and vice versa.

(L1)	$\sigma \cdot (\nu\widetilde{l})\sigma'$	$\preceq_0$	$\sigma \cdot (\nu\widetilde{l})(\alpha \cdot \sigma')$	if $\alpha = \exists ?^{-}$ and $(\nu\widetilde{l})\sigma' \neq UNDEF$
(L2)	$\sigma \cdot (\nu\widetilde{l})(\phi \cdot \exists ?\beta \cdot \sigma')$	$\preceq_0$	$\sigma \cdot (\nu\widetilde{l})(\exists ?\beta \cdot \phi \cdot \sigma')$	if $(\nu\widetilde{l})(\phi \cdot \exists ?\beta \cdot \sigma') \neq UNDEF$
(L3)	$\sigma \cdot (\nu\widetilde{l})\sigma'$	$\preceq_0$	$\sigma \cdot (\nu\widetilde{l})(\exists ?\beta \cdot \beta \cdot \sigma')$	if $(\nu\widetilde{l})\sigma' \neq UNDEF$
(L4)	$\sigma \cdot \langle l' \rangle @ l_1 : l_2 \cdot (\sigma'[^{l'}/_{l}])$	$\preceq_0$	$\sigma \cdot (\nu l)\,\langle l \rangle @ l_1 : l_2 \cdot \sigma'$	
(L5)	$\sigma \cdot \phi \cdot \exists ?l \cdot \sigma'$	$\preceq_0$	$\sigma \cdot \phi \cdot \sigma'$	if $l \in \Upsilon(\phi)$
(L6)	$\sigma \cdot \exists ?l \cdot \beta \cdot \sigma'$	$\preceq_0$	$\sigma \cdot \beta \cdot \exists ?l \cdot \sigma'$	if $l \notin \Upsilon(\beta)$
(L7)	$\sigma \cdot l_1 \curvearrowright l_2 \cdot (\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_1 \cdot \sigma'$	$\preceq_0$	$\sigma \cdot (\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2 \cdot \sigma'$	if $l_1 \neq l_2$
(L8)	$\sigma \cdot \exists ?(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2 \cdot \sigma'$	$\preceq_0$	$\sigma \cdot \exists ?(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_1 \cdot \exists ?l_1 \curvearrowright l_2 \cdot \sigma'$	
(L9)	$\sigma \cdot \exists ?(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_1 \cdot \exists ?l_1 \curvearrowright l_2 \cdot \sigma'$	$\preceq_0$	$\sigma \cdot \exists ?(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2 \cdot \sigma'$	

where, in laws (L5) and (L6), function $\Upsilon(\cdot)$ is defined as follows:

$$\Upsilon(\exists ?l_2) = \{l_2\} \qquad \Upsilon(l_2 \curvearrowright l_1) = \Upsilon(\exists ?l_1 \curvearrowright l_2) = \{l_1, l_2\}$$

$$\Upsilon((\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2) = \Upsilon(\exists ?(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2) = \widetilde{l} \cup \{l_1, l_2\}$$

Table 7: Axioms for the Pre-order Relation on Traces

5.2 Soundness w.r.t. May Testing

To prove that trace equivalence exactly captures may testing, we exploit Definition 3.8; ths, we use OK to denote label $\langle\rangle @$ `test` : `test`, i.e., the action that must be exhibited in any successful computation.

To carry out proofs, we found it convenient to introduce a *complementation function* $\overline{\,\cdot\,}$ over visible labels defined as follows:

$$\begin{array}{rcl@{\qquad}rcl}
\overline{(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2} & \triangleq & \exists ?(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2 & \overline{\exists ?(\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2} & \triangleq & (\nu\widetilde{l})\,\langle l \rangle @ l_1 : l_2 \\
\overline{\exists ?l} & \triangleq & l \curvearrowright l & \overline{\exists ?l_1 \curvearrowright l_2} & \triangleq & l_1 \curvearrowright l_2 \\
\overline{l_1 \curvearrowright l_2} & \triangleq & \begin{cases} \exists ?l_1 \curvearrowright l_2 & \text{if } l_1 \neq l_2 \\ \exists ?l_1 & \text{otherwise} \end{cases} & & &
\end{array}$$

The complementation function is then extended to traces as expected, i.e. $\overline{\epsilon} \triangleq \epsilon$ and $\overline{\phi \cdot \sigma'} \triangleq \overline{\phi} \cdot \overline{\sigma'}$. The usefulness of the complementation function becomes apparent in the following Lemma, that describes a sufficient and a necessary condition for a computation to succeed.

Lemma 5.5 *Let N be a net and O be an observer. Then*

1. *$N \stackrel{\sigma}{\Longrightarrow}$ and $O \stackrel{\overline{\sigma}\cdot \mathtt{OK}}{\Longrightarrow}$ imply that $N \parallel O \stackrel{\mathtt{OK}}{\Longrightarrow}$;*

2. *$N \parallel O \stackrel{\mathtt{OK}}{\Longrightarrow}$ implies that there exists a σ such that $N \stackrel{\sigma}{\Longrightarrow}$ and $O \stackrel{\overline{\sigma}\cdot \mathtt{OK}}{\Longrightarrow}$.*

Proof:

1. The proof is by induction on the length 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 simple; we explicitly present only the case for $\phi = \exists?(\nu l)\ \langle l\rangle @ l_1 : l_2$. By hypothesis, we have that $N \Rightarrow N' \xrightarrow{\exists?(\nu l)\ \langle l\rangle @ l_2 : l_1} N'' \overset{\sigma'}{\Longrightarrow}$, for $l \notin fn(N')$, and $O \Rightarrow O' \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1} O'' \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$; by Proposition 4.2(3), $O' \equiv (\nu l)(O'' \parallel l_2 :: \langle l\rangle \parallel \{l_1 \leftrightarrow l_2\})$. By rule (EXT), $N \parallel O \Rightarrow N' \parallel O' \equiv (\nu l)(N' \parallel O'' \parallel l_2 :: \langle l\rangle \parallel \{l_1 \leftrightarrow l_2\}) \xrightarrow{\tau} (\nu l)(N'' \parallel O'')$. By induction, we have that $N'' \parallel O'' \overset{\mathtt{OK}}{\Longrightarrow}$. Now observe that, by Definition 3.6, O cannot emit `test` in any datum. Hence, $l \neq$ `test` and $(\nu l)(N'' \parallel O'') \overset{\mathtt{OK}}{\Longrightarrow}$; this suffices to conclude.

2. By definition, it must be that $N \parallel O\ (\xrightarrow{\tau}\ )^n H \xrightarrow{\mathtt{OK}}$; the proof is by induction on n. The base step is simple: it suffices to take $\sigma = \epsilon$. For the inductive step, we have that $N \parallel O \xrightarrow{\tau} H'(\xrightarrow{\tau}\ )^{n-1}H$. According to Proposition 5.2, there are ten possibilities for the first τ-step, namely cases 1, 3 – 6 and 9 of such a Proposition, plus the symmetric versions of cases 1, 3, 4 and 9 (that can be handled similarly); indeed, since observers cannot perform remote input actions, the symmetic versions of cases 5 and 6 are meaningless.

Case 1. $N \xrightarrow{\tau} N'$ and $H' \equiv N' \parallel O$: by induction, $N' \overset{\sigma'}{\Longrightarrow}$ and $O \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$, for some σ'. It suffices to take $\sigma = \sigma'$.

Symmetric of case 1. $O \xrightarrow{\tau} O'$ and $H' \equiv N \parallel O'$: analogous.

Case 3. $N \xrightarrow{\exists?l_1 \curvearrowright l_2} N'$, $O \xrightarrow{l_1 \curvearrowright l_2} O'$ and $H' \equiv N' \parallel O'$: by induction, $N' \overset{\sigma'}{\Longrightarrow}$ and $O \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$, for some σ'. It suffices to take $\sigma = \exists?l_1 \curvearrowright l_2 \cdot \sigma'$.

Symmetric of case 3. $N \xrightarrow{l_1 \curvearrowright l_2} N'$, $O \xrightarrow{\exists?l_1 \curvearrowright l_2} O'$ and $H' \equiv N' \parallel O'$: by induction, $N' \overset{\sigma'}{\Longrightarrow}$ and $O' \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$, for some σ'. If $l_1 \neq l_2$, it suffices to take $\sigma = l_1 \curvearrowright l_2 \cdot \sigma'$. Otherwise, by a straightforward induction on the inference for $O \xrightarrow{\exists?l_1 \curvearrowright l_1} O'$, it is easy to prove that l_1 is the address of a node in O; thus, $O \xrightarrow{\tau} O'$. Moreover, by Proposition 4.2(1), it holds that $N \equiv N' \parallel \{l_1 \leftrightarrow l_1\}$; thus, it suffices to take $\sigma = \sigma'$.

Case 4. $N \xrightarrow{\exists?(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_2} N'$, $O \xrightarrow{(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_2} O'$ and $H' \equiv (\nu\widetilde{l})(N' \parallel O')$: by definition of observers, `test` $\notin \widetilde{l}$; thus, $H \equiv (\nu\widetilde{l})H''$ and $N' \parallel O'\ (\xrightarrow{\tau}\ )^{n-1}H'' \xrightarrow{\mathtt{OK}}$. By induction, $N' \overset{\sigma'}{\Longrightarrow}$ and $O' \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$, for some σ'. Thus, it suffices to take $\sigma = \exists?(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_2 \cdot \sigma'$.

Symmetric of case 4. $N \xrightarrow{(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_1} N'$, $O \xrightarrow{\exists?(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_1} O'$ and $H' \equiv (\nu\widetilde{l})(N' \parallel O')$: similarly to the previous case, it suffices to take $\sigma = (\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_1 \cdot \sigma'$, where σ' is the trace returned by the inductive hypothesis. Notice that in this case the more general label $\exists?(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_2$ (for $l_1 \neq l_2$) is not necessary, since observers can only perform local inputs.

Case 5. $N \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists?(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_1} N'$, $O \xrightarrow{(\nu\widetilde{l})\ \langle l\rangle @ l_2 : l_2} O'$ and $H' \equiv (\nu\widetilde{l})(N' \parallel O')$: by induction, there exists a σ' such that $N' \overset{\sigma'}{\Longrightarrow}$ and $O' \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$. Notice that $\widetilde{l} \cap fn(N) = \emptyset$, since we assumed that bound names are different from the free ones; hence, by rules (LTS-BIN) and (LTS-DATUMREQ), it holds that $N \xrightarrow{\exists?(\nu\widetilde{l})\ \langle l\rangle @ l_2 : l_2} N'$. Now, take $\sigma = \exists?(\nu\widetilde{l})\ \langle l\rangle @ l_2 : l_2 \cdot \sigma'$ and easily conclude.

Case 6. $N \xrightarrow{(\nu\widetilde{l})\,\langle l\rangle @ l_2 : l_2} \xrightarrow{\exists?\,\langle l\rangle @ l_2 : l_1} N'$, $O \xrightarrow{l_2 \curvearrowright l_1} O'$ and $H' \equiv (\nu\widetilde{l})(N' \parallel O')$: this case is similar to the previous one, by exploiting rule (LTS-CONNREQ) and taking $\sigma = \exists?l_2 \curvearrowright l_1 \cdot \sigma'$.

Case 9. $N \xrightarrow{\exists?l} N'$, $O \xrightarrow{l \curvearrowright l} O'$ and $H' \equiv N' \parallel O'$: this case simply follows by taking $\sigma = l \curvearrowright l \cdot \sigma'$, where σ' is the trace from the induction hypothesis.

Symmetric of case 9. $N \xrightarrow{l \curvearrowright l} N'$, $O \xrightarrow{\exists?l} O'$ and $H' \equiv N' \parallel O'$: like the previous case, with $\sigma = \exists?l\ \cdot \sigma'$. ∎

The next Lemma states that, if an observer can observe a trace σ (i.e., can perform $\overline{\sigma}$), then it can also observe any trace $\sigma' \leq \sigma$.

Lemma 5.6 *If $\sigma' \leq \sigma$ and $O \overset{\overline{\sigma}}{\Longrightarrow}$, then $O \overset{\overline{\sigma'}}{\Longrightarrow}$.*

Proof: By definition, $\sigma' \leq \sigma$ means $\sigma'(\leq_0)^n\sigma$ for some $n \geq 0$; 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 $O \overset{\overline{\sigma}}{\Longrightarrow}$ implies that $O \overset{\overline{\sigma''}}{\Longrightarrow}$. Indeed, by induction, the latter judgement implies that $O \overset{\overline{\sigma'}}{\Longrightarrow}$, as required. We reason by case analysis on the law in Table 7 used to infer $\sigma'' \leq_0 \sigma$. Notably, since observers cannot perform remote inputs, law (L7) cannot be used to infer $\sigma'' \leq_0 \sigma$; moreover, in law (L3), only local inputs are allowed.

(L1).a: $\sigma \triangleq \sigma_1 \cdot \exists?(\nu\widetilde{l})\ \langle l\rangle @ l_2\ :\ l_1\ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu\widetilde{l})\sigma_2$. By hypothesis, $O \overset{\overline{\sigma_1}}{\Longrightarrow} O' \xrightarrow{(\nu\widetilde{l})\,\langle l\rangle @ l_2 : l_1} O'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; by Proposition 4.2(2/3), $O' \equiv (\nu\widetilde{l})(O'' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle)$. Now, if $\widetilde{l} = \emptyset$ or $\widetilde{l} \cap fn(\sigma_2) = \emptyset$, then $O' \overset{\overline{\sigma_2}}{\Longrightarrow}$ and, hence, $O \overset{\overline{\sigma''}}{\Longrightarrow}$. Otherwise, it must be $\widetilde{l} = \{l\}$ and $\sigma_2 \triangleq \sigma_3 \cdot \exists?\,\langle l\rangle @ l_4\ :\ l_3\ \cdot\sigma_4$, for $l \notin fn(\sigma_3, l_3, l_4)$; thus, $O'' \overset{\overline{\sigma_3}}{\Longrightarrow} O''_1 \xrightarrow{\langle l\rangle @ l_4 : l_3} O''_2 \overset{\overline{\sigma_4}}{\Longrightarrow}$. Now, $O' \overset{\overline{\sigma_3}}{\Longrightarrow} (\nu l)(O''_2 \parallel \{l_3 \leftrightarrow l_4\} \parallel l_4 :: \langle l\rangle \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle)$. Hence, $O \xRightarrow{\overline{\sigma_1}\cdot\overline{\sigma_3}\cdot(\nu l)\ \langle l\rangle @ l_4 : l_3\ \cdot\overline{\sigma_4}}$, i.e. $O \overset{\overline{\sigma''}}{\Longrightarrow}$.

(L1).b: $\sigma \triangleq \sigma_1 \cdot \exists?l_1 \curvearrowright l_2 \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \sigma_2$. By hypothesis, $O \overset{\overline{\sigma_1}}{\Longrightarrow} O' \xrightarrow{l_1 \curvearrowright l_2} O'' \overset{\overline{\sigma_2}}{\Longrightarrow}$; by Proposition 4.2(1), $O' \equiv O'' \parallel \{l_1 \leftrightarrow l_2\}$ and hence $O \xRightarrow{\overline{\sigma_1}\cdot\overline{\sigma_2}}$, as required.

(L1).c: $\sigma \triangleq \sigma_1 \cdot \exists?l\ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \sigma_2$. Similar to the previous sub-case.

(L2).a: $\sigma \triangleq \sigma_1 \cdot \exists?(\nu\widetilde{l})\,\langle l\rangle @ l_2\ :\ l_1\ \cdot\phi\cdot\sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu\widetilde{l})(\phi \cdot \exists?\,\langle l\rangle @ l_2\ :\ l_1\ \cdot\sigma_2)$. By hypothesis, $O \overset{\overline{\sigma_1}}{\Longrightarrow} O' \xrightarrow{(\nu\widetilde{l})\,\langle l\rangle @ l_2 : l_1} O'' \overset{\overline{\phi}}{\Rightarrow} O''' \overset{\overline{\sigma_2}}{\Longrightarrow}$; by Proposition 4.2(2/3), $O' \equiv (\nu\widetilde{l})(O'' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle)$. Now, if $\widetilde{l} = \emptyset$ or $\widetilde{l} \cap fn(\phi) = \emptyset$, then it must be that $O' \xRightarrow{\overline{\phi}\cdot(\nu\widetilde{l})\ \langle l\rangle @ l_2 : l_1\ \cdot\overline{\sigma_2}}$ and, hence, $O \overset{\overline{\sigma''}}{\Longrightarrow}$. Otherwise, it must be $\phi = \exists?\,\langle l\rangle @ l_4\ :\ l_3$ for $l \notin \{l_3, l_4\}$; thus, $O'' \xRightarrow{\langle l\rangle @ l_4 : l_3} O'''$. Now, $O' \xRightarrow{(\nu l)\ \langle l\rangle @ l_4 : l_3} O''' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \xrightarrow{\langle l\rangle @ l_2 : l_1} O''' \overset{\overline{\sigma_2}}{\Longrightarrow}$, and hence $O \overset{\overline{\sigma''}}{\Longrightarrow}$.

(L2).b: $\sigma \triangleq \sigma_1 \cdot \exists?l_1 \curvearrowright l_2 \cdot \phi \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \phi \cdot \exists?l_1 \curvearrowright l_2 \cdot \sigma_2$. By hypothesis, $O \overset{\overline{\sigma_1}}{\Longrightarrow} O' \xrightarrow{l_1 \curvearrowright l_2} O'' \overset{\overline{\phi}}{\Rightarrow} O''' \overset{\overline{\sigma_2}}{\Longrightarrow}$ and, by Proposition 4.2(1), $O' \equiv O'' \parallel \{l_1 \leftrightarrow l_2\}$. This implies that $O \xRightarrow{\overline{\sigma_1}\cdot\overline{\phi}\cdot l_1 \curvearrowright l_2 \cdot\overline{\sigma_2}}$, as required.

(L3).a: $\sigma \triangleq \sigma_1 \cdot \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1 \ \cdot \langle l \rangle @ l_1 : l_1 \ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu\widetilde{l})\sigma_2$. By hypothesis, $O \xRightarrow{\overline{\sigma_1}} O' \xrightarrow{(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1} O'_1 \Rightarrow O'_2 \xrightarrow{\exists ?\ \langle l \rangle @ l_1 : l_1} O'' \xRightarrow{\overline{\sigma_2}}$; moreover, by Proposition 4.2(2/3), $O' \equiv (\nu\widetilde{l})(O'_1 \parallel l_1 :: \langle l \rangle)$. Thus, $O' \Rightarrow (\nu\widetilde{l})(O'_2 \parallel l_1 :: \langle l \rangle) \xrightarrow{\tau} (\nu\widetilde{l})O''$. Now, if $\widetilde{l} = \emptyset$ or $\widetilde{l} \cap fn(\sigma_2) = \emptyset$, then $(\nu\widetilde{l})O'' \xRightarrow{\overline{\sigma_2}}$ and, hence, $O \xRightarrow{\overline{\sigma''}}$. Otherwise, we reason as in case **(L1).a** to obtain that $(\nu\widetilde{l})O'' \xRightarrow{\overline{(\nu\widetilde{l})\sigma_2}}$ and, again, $O \xRightarrow{\overline{\sigma''}}$.

(L3).b: $\sigma \triangleq \sigma_1 \cdot \exists ? l_1 \curvearrowright l_2 \cdot l_1 \curvearrowright l_2 \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \sigma_2$. By hypothesis, $O \xRightarrow{\overline{\sigma_1}} O' \xrightarrow{l_1 \curvearrowright l_2} O'' \xRightarrow{\exists ? l_1 \curvearrowright l_2} O''' \xRightarrow{\overline{\sigma_2}}$; moreover, by Proposition 4.2(1), $O' \equiv O'' \parallel \{l_1 \leftrightarrow l_2\}$. By exploiting (LTS-COMPL), this implies that $O' \Rightarrow O'''$ and we can easily conclude.

(L4): trivial, since we are using an early-style LTS.

(L5): The cases for $\phi = \exists ?\text{-}$ are simple and can be dealt with by relying on Proposition 4.2. Let us consider the case for $\phi = (\nu l)\ \langle l \rangle @ l_1 : l_1$, since the case for $l_1 \curvearrowright l_2$ is similar; thus, $\sigma \triangleq \sigma_1 \cdot (\nu l)\ \langle l \rangle @ l_1 : l_1 \ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot (\nu l)\ \langle l \rangle @ l_1 : l_1 \ \cdot \exists ? l' \curvearrowright l' \cdot \sigma_2$, for $l' \in \{l, l_1\}$. By hypothesis, $O \xRightarrow{\overline{\sigma_1}} O' \xrightarrow{\exists ?(\nu l)\ \langle l \rangle @ l_1 : l_1} O'' \xRightarrow{\overline{\sigma_2}}$; now, by definition of the LTS, it is simple to show that l' is the address of a node in O'' and hence $O'' \equiv O'' \parallel l' :: \mathbf{nil}$. Thus, by exploiting rule (SELF), we have that $O \xRightarrow{\overline{\sigma_1} \cdot \exists ?(\nu l)\ \langle l \rangle @ l_1 : l_1} O'' \parallel l' :: \mathbf{nil} \xrightarrow{l' \curvearrowright l'} O'' \xRightarrow{\overline{\sigma_2}}$, as required.

(L6): $\sigma \triangleq \sigma_1 \cdot \beta \cdot \exists ? l \curvearrowright l \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \exists ? l \curvearrowright l \cdot \beta \cdot \sigma_2$. By hypothesis, $O \xRightarrow{\overline{\sigma_1}} O_1 \xrightarrow{\overline{\beta}} O_2 \Rightarrow O_3 \xrightarrow{l \curvearrowright l} O_4 \xRightarrow{\overline{\sigma_2}}$; moreover, by Proposition 4.2(1), l is the address of a node in O_3. By definition of the LTS, it is easy to see that τ-actions cannot change the adresses of a net, while $\overline{\beta}$ at most introduces nodes with address in $\Upsilon(\beta)$. Since $l \notin \Upsilon(\beta)$, it follows that l is the address of a node in O_1; hence, $O \xRightarrow{\overline{\sigma_1} \cdot l \curvearrowright l \cdot \overline{\beta} \cdot \overline{\sigma_2}}$, as required.

(L8): $\sigma \triangleq \sigma_1 \cdot \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1 \ \cdot \exists ? l_1 \curvearrowright l_2 \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2 \ \cdot \sigma_2$. By hypothesis, $O \xRightarrow{\overline{\sigma_1}} O_1 \xrightarrow{(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1} O_2 \Rightarrow O_3 \xrightarrow{l_1 \curvearrowright l_2} O_4 \xRightarrow{\overline{\sigma_2}}$; by Proposition 4.2(2/3), $O_1 \equiv (\nu\widetilde{l})(O_2 \parallel l_1 :: \langle l \rangle)$ and $O_3 \equiv O_4 \parallel \{l_1 \leftrightarrow l_2\}$. Thus, $O_1 \Rightarrow (\nu\widetilde{l})(O_4 \parallel \{l_1 \leftrightarrow l_2\} \parallel l_1 :: \langle l \rangle)$ and we can easily conclude that $O \xRightarrow{\overline{\sigma_1} \cdot (\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2 \ \cdot \overline{\sigma_2}}$, as required.

(L9): $\sigma \triangleq \sigma_1 \cdot \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2 \ \cdot \sigma_2$ and $\sigma'' \triangleq \sigma_1 \cdot \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1 \ \cdot \exists ? l_1 \curvearrowright l_2 \cdot \sigma_2$. By hypothesis, $O \xRightarrow{\overline{\sigma_1}} O' \xrightarrow{(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2} O'' \xRightarrow{\overline{\sigma_2}}$; by Proposition 4.2(2/3), $O' \equiv (\nu\widetilde{l})(O'' \parallel l_1 :: \langle l \rangle \parallel \{l_1 \leftrightarrow l_2\})$ and we can easily conclude that $O \xRightarrow{\overline{\sigma_1} \cdot (\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1 \ \cdot l_1 \curvearrowright l_2 \cdot \overline{\sigma_2}}$, as required. ∎

By properly adapting the proof of the previous Lemma, we can also prove that the laws in Table 7 are 'sound', in the sense that, whenever $\sigma' \preceq_0 \sigma$ and $N \xRightarrow{\sigma'}$, any net M able to perform $\overline{\sigma}$ may synchronise with N.

Proposition 5.7 *Let* $\sigma' \preceq_0 \sigma$, $N \xRightarrow{\sigma'}$ *and* $M \xRightarrow{\overline{\sigma} \cdot \phi}$*; then,* $N \parallel M \xRightarrow{\phi}$.

Proof: Let us first assume that $\sigma' \preceq_0 \sigma$ has been inferred by using law (L7). In this case, $\sigma \triangleq \sigma_1 \cdot (\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2 \ \cdot \sigma_2$ and $\sigma' \triangleq \sigma_1 \cdot l_1 \curvearrowright l_2 \cdot (\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1 \ \cdot \sigma_2$; thus, $M \xRightarrow{\overline{\sigma_1} \cdot \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2 \ \cdot \overline{\sigma_2} \cdot \phi}$ and $N \xRightarrow{\sigma_1} N_1 \xrightarrow{l_1 \curvearrowright l_2} N_2 \Rightarrow N_3 \xrightarrow{(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1} N_4 \xRightarrow{\sigma_2} N'$. By Proposition 4.2, $N \xRightarrow{\sigma_1} N_2 \parallel \{l_1 \leftrightarrow$

$l_2\}\Rightarrow(\nu\widetilde{l})(N_4 \parallel l_1 :: \langle l\rangle \parallel \{l_1 \leftrightarrow l_2\}) \overset{\sigma_2}{\Longrightarrow}$; hence, by (LTS-OFFER), $N \overset{\sigma_1}{\Longrightarrow} (\nu\widetilde{l})(N_4 \parallel l_1 :: \langle l\rangle \parallel \{l_1 \leftrightarrow l_2\}) \xrightarrow{(\nu\widetilde{l})\,\langle l\rangle\,@\,l_1:l_2} N_4 \overset{\sigma_2}{\Longrightarrow} N'$ and, by properly adapting Lemma 5.5, this implies that $N \parallel M \overset{\phi}{\Longrightarrow}$.

If $\sigma' \preceq_0 \sigma$ has been inferred by using the remaining laws, we can easily adapt the proof of Lemma 5.6 with M in place of O (just notice that case **(**L3**).a** can be smoothly adapted to also consider a remote input); this suffi ces to conclude. ∎

Now, the main theorem of this section follows.

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

Proof: Let O be an observer such that $N \parallel O \overset{\mathtt{OK}}{\Longrightarrow}$. By Lemma 5.5(2), there exists σ such that $N \overset{\sigma}{\Longrightarrow}$ and $O \overset{\overline{\sigma}\cdot\mathtt{OK}}{\Longrightarrow}$. By Defi nition 5.4, there exists $\sigma' \preceq \sigma$ such that $M \overset{\sigma'}{\Longrightarrow}$; by suffi x closure of $\preceq$ (that can be easily proved), we have that $\sigma' \cdot \exists?\mathtt{OK} \preceq \sigma \cdot \exists?\mathtt{OK}$. So, by Lemma 5.6, $O \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$ that, Lemma 5.5(1), implies that $M \parallel O \overset{\mathtt{OK}}{\Longrightarrow}$, as required by Defi nition 3.8. ∎

5.3 Completeness w.r.t. May Testing

To prove that trace equivalence exactly captures may testing, we defi ne a family of observers as follows.

Definition 5.9 *Given a trace σ, the* canonical observer *for σ, written $\amalg(\sigma)$, is*

$$\amalg(\sigma) \;=\; N \parallel \mathtt{test} :: P$$

where the actual observer process P and net N enabling the observation are returned by $O_\emptyset(\sigma) = \,<P\,;N>$, which is defined in Table 8.

In Table 8, we write $\lceil l :: \mathbf{nil} \rfloor_L$ to denote $l :: \mathbf{nil}$, if $l \notin L$, and $\mathbf{0}$, otherwise. With abuse of notation, we use process GO l DO _ THEN P introduced in Section 4.2 also when _ is a sequence of actions. In $O_L(\sigma)$, L is the (fi nite) 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. In the pair $<P\,;N>$, the net N has only to provide nodes in order to enable P's observations. Indeed, N is just a parallel of nodes hosting the inert process and its traces can only be sequences of labels of the form $l \curvearrowright l$. However, N must not provide a node with locality l whenever $l \in L$. In this case, the observed net already provides the needed node: indeed, if $l \in L$, then l has been extruded by an action $(\nu l)\,\langle l\rangle\,@\,l_1 : l_2$ in σ.

The key property of the canonical observer for σ is that it always yields a successful computation when run in parallel with a net that may perform σ, as stated by the following Proposition.

Proposition 5.10 *If $M \overset{\sigma}{\Longrightarrow}$, then $M \parallel \amalg(\sigma) \overset{\mathtt{OK}}{\Longrightarrow}$.*

Proof: The proof is by induction on $|\sigma|$. The base step is trivial. The inductive step follows from Defi nition 5.9; there are only two non-trivial cases:

$\sigma = (\nu\widetilde{l})\,\langle l\rangle\,@\,l_1 : l_2 \cdot \sigma'$**.** In this case, $M \Rightarrow M' \xrightarrow{(\nu\widetilde{l})\,\langle l\rangle\,@\,l_1:l_2} M'' \overset{\sigma'}{\Longrightarrow}$; by Proposition 4.2(2/3), $M' \equiv (\nu\widetilde{l})(M'' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_1 :: \langle l\rangle)$. Moreover, by construction, $\amalg(\sigma) \xRightarrow{\exists?l_1\curvearrowright l_2\cdot\exists?(\nu\widetilde{l})\,\langle l\rangle\,@\,l_1:l_1} \amalg(\sigma')$; hence, $M \parallel \amalg(\sigma) \Rightarrow (\nu\widetilde{l})(M'' \parallel \amalg(\sigma'))$ and, by induction, we easily conclude.

$$\begin{aligned}
O_L(\epsilon) &= \; <\mathbf{out}()@\texttt{test}.\mathbf{nil}\,;\mathbf{0}> \\
O_L(l_1 \curvearrowright l_2 \cdot \sigma) &= \; <\textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2)\ \textsc{Then}\ P\,;N \parallel \lceil l_1 :: \mathbf{nil}\rfloor_L> \\
&\quad \text{where } O_L(\sigma) = \; <P\,;N> \\
O_L(\langle l\rangle @ l_1 : l_2 \cdot \sigma) &= \; <\textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2).\mathbf{in}(l)@l_1\ \textsc{Then}\ P\,;N \parallel \lceil l_1 :: \mathbf{nil}\rfloor_L> \\
&\quad \text{where } O_L(\sigma) = \; <P\,;N> \\
O_L((\nu l)\,\langle l\rangle @ l_1 : l_2 \cdot \sigma) &= \; <\textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2).\mathbf{in}(!x)@l_1\ \textsc{Then}\ (P[{}^{x}\!/\!_{l}])\,;N \parallel \lceil l_1 :: \mathbf{nil}\rfloor_L> \\
&\quad \text{where } O_{L\cup\{l\}}(\sigma) = \; <P\,;N> \\
O_L(\exists ?l \cdot \sigma) &= \; <P\,;N \parallel \lceil l :: \mathbf{nil}\rfloor_L> \\
&\quad \text{where } O_L(\sigma) = \; <P\,;N> \\
O_L(\exists ?l_1 \curvearrowright l_2 \cdot \sigma) &= \; <\textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{conn}(l_2)\ \textsc{Then}\ P\,;N \parallel \lceil l_1 :: \mathbf{nil}\rfloor_L \parallel \lceil l_2 :: \mathbf{nil}\rfloor_L> \\
&\quad \text{where } O_L(\sigma) = \; <P\,;N> \\
O_L(\exists ?\,\langle l\rangle @ l_2 : l_1 \cdot \sigma) &= \; <\textsc{Go}\ l_2\ \textsc{Do}\ \mathbf{conn}(l_1).\mathbf{out}(l)@l_2\ \textsc{Then}\ P\ ; \\
&\qquad N \parallel \lceil l_1 :: \mathbf{nil}\rfloor_L \parallel \lceil l_2 :: \mathbf{nil}\rfloor_L> \\
&\quad \text{where } O_L(\sigma) = \; <P\,;N> \\
O_L(\exists ?(\nu l)\,\langle l\rangle @ l_2 : l_1 \cdot \sigma) &= \; <\textsc{Go}\ l_2\ \textsc{Do}\ \mathbf{conn}(l_1).\mathbf{new}(l).\mathbf{out}(l)@l_2\ \textsc{Then}\ P\ ; \\
&\qquad N \parallel \lceil l_1 :: \mathbf{nil}\rfloor_L \parallel \lceil l_2 :: \mathbf{nil}\rfloor_L> \\
&\quad \text{where } O_L(\sigma) = \; <P\,;N>
\end{aligned}$$

Table 8: Constructing the Canonical Observers for Theorem 5.13

$\sigma = l \curvearrowright l \cdot \sigma'$. In this case, $\amalg(\sigma)$ does not necessarily exhibit label $\exists ?l$ to become $\amalg(\sigma')$ (while, by definition, it will surely exhibit label $\exists ?l \curvearrowright l$). However, by Proposition 4.2(1), $M \equiv M \parallel l :: \mathbf{nil} \overset{\sigma'}{\Longrightarrow}$ and $\amalg(\sigma) \triangleq N \parallel l :: \mathbf{nil} \parallel \texttt{test} :: \textsc{Go}\ l\ \textsc{Do}\ \mathbf{disc}(l)\ \textsc{Then}\ P \Rightarrow N \parallel l :: \mathbf{nil} \parallel \texttt{test} :: P \triangleq \amalg(\sigma') \parallel l :: \mathbf{nil}$, for $<P\,;N> \; = O_\emptyset(\sigma')$. By induction, $M \parallel \amalg(\sigma') \overset{\texttt{OK}}{\Longrightarrow}$; hence, $M \parallel \amalg(\sigma) \Rightarrow M \parallel \amalg(\sigma') \overset{\texttt{OK}}{\Longrightarrow}$. ∎

The next lemma states that, if $\amalg(\sigma)$ reports success by interacting with a net N, then it can do so by performing a trace $\overline{\sigma'}$ that does not contain the reserved name `test` and labels of the form $\exists ?l$ (that require existence of node l and are generated when an action **conn**is executed). Due to the definition of the canonical observers (and to notation $\textsc{Go}\ l\ \textsc{Do}\ a\ \textsc{Then}\ P$), this means that all labels in $\overline{\sigma'}$ are from $\overline{\sigma}$ or are generated by the localities in the observation context (the latter labels are of the form $l \curvearrowright l$).

Lemma 5.11 *If $M \parallel \amalg(\sigma) \overset{\texttt{OK}}{\Longrightarrow}$, then there exists a σ' such that $M \overset{\sigma'}{\Longrightarrow}$, $\amalg(\sigma) \overset{\overline{\sigma'}\cdot\texttt{OK}}{\Longrightarrow}$, $\texttt{test} \notin n(\sigma')$ and $\overline{\sigma'}$ does not contain labels of the form $\exists ?l$.*

Proof: By Lemma 5.5(2), we know that there exists a trace σ'' such that $M \overset{\sigma''}{\Longrightarrow}$ and $\amalg(\sigma) \overset{\overline{\sigma''}\cdot\texttt{OK}}{\Longrightarrow}$; moreover, since `test` is a reserved name, $\texttt{test} \notin n(\sigma'')$. The proof now proceeds by induction on the number of labels of the form $\exists ?l$ in $\overline{\sigma''}$. The base step is trivial. For the inductive step, suppose that $\overline{\sigma''}$ is of the form $\overline{\sigma_1} \cdot \phi \cdot \overline{\sigma_2}$, where ϕ is the first label of kind $\exists?l$ in $\overline{\sigma''}$. Let $\phi = \exists?l_2$ and L be the set of names extruded by σ_1; we consider two sub-cases:

1. $l_2 \notin L$. By definition of canonical observers, it must be that $\amalg(\sigma) \xRightarrow{\overline{\sigma_1}} N \parallel l_1 :: \mathbf{nil} \parallel l_2 :: \mathbf{nil} \parallel l_1 :: \mathbf{conn}(l_2).P \parallel \{\mathtt{test} \leftrightarrow l_1\} \xrightarrow{\exists?l_2} N \parallel l_1 :: \mathbf{nil} \parallel l_2 :: \mathbf{nil} \parallel l_1 :: P \parallel \{\mathtt{test} \leftrightarrow l_1\} \xRightarrow{\overline{\sigma_2}\cdot\mathtt{OK}}$, for proper N and P; thus, we can easily conclude that $\amalg(\sigma) \xRightarrow{\overline{\sigma_1}\cdot\overline{\sigma_2}\cdot\mathtt{OK}}$. Additionally, by hypothesis, $M \xRightarrow{\sigma_1} M' \xrightarrow{l_2 \curvearrowright l_2} M'' \xRightarrow{\sigma_2}$ and so $M' \equiv M'' \parallel \{l_2 \leftrightarrow l_2\}$; again, we can easily conclude that $M \xRightarrow{\sigma_1\cdot\sigma_2}$. The thesis follows from the inductive hypothesis on $\overline{\sigma_1} \cdot \overline{\sigma_2}$.

2. $l_2 \in L$. Then, σ_1 is of the form $\sigma_3 \cdot (\nu l)\,\langle l_2\rangle\,@\,l_3 : l_4 \cdot \sigma_4$; thus, by rule (LTS-BIN), it must be that $\amalg(\sigma) \xRightarrow{\overline{\sigma_3}} K \xrightarrow{\exists?(\nu l_2)\,\langle l_2\rangle\,@\,l_3:\,l_4} K' \parallel l_2 :: \mathbf{nil} \xRightarrow{\overline{\sigma_4}} N \parallel l_1 :: \mathbf{nil} \parallel l_2 :: \mathbf{nil} \parallel l_1 :: \mathbf{conn}(l_2).P \parallel \{\mathtt{test} \leftrightarrow l_1\} \xRightarrow{\exists?l\,\cdot\overline{\sigma_2}\cdot\mathtt{OK}}$ and the proof proceeds like in the previous case. ∎

The main Lemma to prove completeness of trace equivalence w.r.t. may testing is the following one: it states that, if $\amalg(\sigma)$ can report success upon execution of a trace $\overline{\sigma'}$ that does not contain the reserved name `test` and labels of the form $\exists?l$, then $\sigma' \preceq \sigma$. The previous Lemma showed that these assumptions on $\overline{\sigma'}$ are not restrictive.

Lemma 5.12 *Let* $\amalg(\sigma) \xRightarrow{\overline{\sigma'}\,\cdot\,\mathtt{OK}}$. *If* `test` $\notin n(\sigma')$ *and* $\overline{\sigma'}$ *does not contain labels of the form* $\exists?l$ *, then* $\sigma' \preceq \sigma$*.*

Proof: The proof is by induction on $|\sigma|$. The base step is trivial. For the inductive step, let σ be $\phi \cdot \sigma''$; we reason by analysis on the cases for ϕ. In what follows, we use notation $\sigma_2\backslash^N_L$ to denote the trace obtained from σ_2 by removing all the labels of the form $l \curvearrowright l$, for every $l \in L$ such that $l \notin fn(N)$.

(i) $\phi \triangleq \exists?l$ **.** By costruction, $\amalg(\sigma) \triangleq N \parallel l :: \mathbf{nil} \parallel \mathtt{test} :: P$, where $< P\,; N > \; = O_\emptyset(\sigma'')$. Now, we have that $\overline{\sigma'} \triangleq \overline{\sigma_1} \cdot \overline{\sigma_2}$, where $N \xRightarrow{\overline{\sigma_1}} N'$ and $N' \parallel l :: \mathbf{nil} \parallel \mathtt{test} :: P \xRightarrow{\overline{\sigma_2}\cdot OK}$. Thus, $\amalg(\sigma'') \triangleq N \parallel \mathtt{test} :: P \xRightarrow{\overline{\sigma_1}\cdot\overline{\sigma'_2}\cdot OK}$, where $\overline{\sigma'_2} = \overline{\sigma_2}\backslash^N_{\{l\}}$. Indeed, by definition of $\amalg(\sigma)$, we have that P does not need $l :: \mathbf{nil}$ to evolve; hence, the only possible contribution of $l :: \mathbf{nil}$ to the production of σ' is by providing (possibly several times) label $l \curvearrowright l$. By induction, we have that $\sigma_1 \cdot \sigma'_2 \preceq \sigma''$. We now have that $\sigma \succeq \phi \cdot \sigma_1 \cdot \sigma'_2 \succeq \phi \cdot \sigma_1 \cdot \sigma_2 \succeq \sigma_1 \cdot \sigma_2 \triangleq \sigma'$, where the first inequality has been obtained by prefix closure, the second inequality has been obtained by repeated applications of laws (L5) and (L2) (as many times as the number of labels $l \curvearrowright l$ removed from $\overline{\sigma_2}$ to obtain $\overline{\sigma'_2}$) and the last inequality relies on law (L1).

(ii) $\phi \triangleq l_1 \curvearrowright l_2$**.** By construction, $\amalg(\sigma) \triangleq N \parallel l_1 :: \mathbf{nil} \parallel \mathtt{test} :: \textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2)\ \textsc{Then}\ P$, where $< P\,; N > \; = O_\emptyset(\sigma'')$. Then, $\overline{\sigma'} \triangleq \overline{\sigma_1} \cdot \exists?l_1 \curvearrowright l_2 \cdot \overline{\sigma_2}$, where $N \xRightarrow{\overline{\sigma_1}} N'$ and $N' \parallel l_1 :: \mathbf{nil} \parallel l_2 :: \mathbf{nil} \parallel \mathtt{test} :: P \xRightarrow{\overline{\sigma_2}\cdot\mathtt{OK}}$. By induction, $\sigma_1 \cdot \sigma'_2 \preceq \sigma''$, where $\overline{\sigma'_2} = \overline{\sigma_2}\backslash^N_{\{l_1,l_2\}}$; we can conclude by prefix closure, by repeated applications of laws (L5) and (L2) (as many times as the number of labels $l_1 \curvearrowright l_1$ and $l_2 \curvearrowright l_2$ removed from $\overline{\sigma_2}$ to obtain $\overline{\sigma'_2}$) and by law (L6); indeed, since N is just a parallel of nodes hosting process **nil**, σ_1 can only consist of labels of kind $\exists?l \curvearrowright l$.

(iii) $\phi \triangleq (\nu\widetilde{l})\,\langle l\rangle\,@\,l_1 : l_2$ **.** By construction, we have two subcases.

1. If $\widetilde{l} = \emptyset$, then $\amalg(\sigma) \triangleq N \parallel l_1 :: \mathbf{nil} \parallel \mathtt{test} :: \textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2).\mathbf{in}(l)@l_1\ \textsc{Then}\ P$, where $< P\,; N > \ = O_{\emptyset}(\sigma'')$. Then, $\overline{\sigma'} \triangleq \overline{\sigma_1} \cdot \exists ?l_1 \curvearrowright l_2 \cdot \exists ?\ \langle l \rangle @ l_1 : l_1\ \cdot \overline{\sigma_2}$, where $N \overset{\overline{\sigma_1}}{\Longrightarrow} N'$ and $N' \parallel l_1 :: \mathbf{nil} \parallel l_2 :: \mathbf{nil} \parallel \mathtt{test} :: P \overset{\overline{\sigma_2}\cdot\mathtt{OK}}{\Longrightarrow}$. By induction, $\sigma_1 \cdot \sigma_2' \preceq \sigma''$, where $\overline{\sigma_2'} = \overline{\sigma_2}\backslash^{N}_{\{l_1,l_2\}}$; similarly to case **(ii)**, we can conclude by prefix closure, by repeated applications of laws (L5) and (L2), by law (L6) and by law (L7).
2. If $\widetilde{l} = \{l\}$, then $\amalg(\sigma) \triangleq N \parallel l_1 :: \mathbf{nil} \parallel \mathtt{test} :: \textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2).\mathbf{in}(!x)@l_1\ \textsc{Then}\ P[^{x}/_{l}]$, where $< P\,; N > \ = O_{\{l\}}(\sigma'')$. We then proceed similarly to case **(iii)**.1, by also exploiting rule (L4) if a name different from l is received. Just notice that, because of rule (LTS-BIn), now also the parallel component $l :: \mathbf{nil}$ may contribute to the generation of σ_2 with labels $l \curvearrowright l$; nevertheless, by definition of function $\Upsilon(\cdot)$, law (L5) can properly handle also this situation.

(iv) $\phi \triangleq \exists ?l_1 \curvearrowright l_2$**.** By construction, $\amalg(\sigma) \triangleq N \parallel l_1 :: \mathbf{nil} \parallel l_2 :: \mathbf{nil} \parallel \mathtt{test} :: \textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{conn}(l_2)\ \textsc{Then}\ P$, where $< P\,; N > \ = O_{\emptyset}(\sigma'')$. Then, $\overline{\sigma'} = \overline{\sigma_1} \cdot \overline{\sigma_2}$, where $N \overset{\sigma_1}{\Longrightarrow} N'$ and $N' \parallel \mathtt{test} :: P \parallel \{l_1 \leftrightarrow l_2\} \overset{\overline{\sigma_2}\cdot\mathtt{OK}}{\Longrightarrow}$. Notice that the component $\{l_1 \leftrightarrow l_2\}$ can contribute to the generation of $\overline{\sigma_2}$ or not. If it does not contribute, the thesis follows by an easy induction, prefix closure and law (L1); otherwise, by exploiting Proposition 5.2 and because observers cannot perform remote inputs, there are only three possible kinds of contributions (for the sake of simplicity, we ignore here labels of kind $l \curvearrowright l$, for $l \in \{l_1, l_2\}$, that can be handled like in case **(ii)**.1):

1. *symmetric version of Proposition 5.2(1):* in this case, $\overline{\sigma_2} \triangleq \overline{\sigma_3} \cdot l_1 \curvearrowright l_2 \cdot \overline{\sigma_4}$, where $N' \parallel \mathtt{test} :: P \overset{\overline{\sigma_3}\cdot\overline{\sigma_4}\cdot\mathtt{OK}}{\Longrightarrow}$. By induction, $\sigma_1 \cdot \sigma_3 \cdot \sigma_4 \preceq \sigma''$; we can conclude by prefix closure and law (L2).
2. *Proposition 5.2(2):* in this case, $\overline{\sigma_2} \triangleq \overline{\sigma_3} \cdot (\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2\ \cdot \overline{\sigma_4}$, where $N' \parallel \mathtt{test} :: P \overset{\overline{\sigma_3}\cdot(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1\ \cdot\overline{\sigma_4}\cdot\mathtt{OK}}{\Longrightarrow}$. By induction, $\sigma_1 \cdot \sigma_3 \cdot \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_1\ \cdot \sigma_4 \preceq \sigma''$; we can conclude by prefix closure and laws (L2) and (L8).
3. *Proposition 5.2(3):* in this case, $\overline{\sigma_2} \triangleq \overline{\sigma_3} \cdot \overline{\sigma_4}$, where $N' \parallel \mathtt{test} :: P \overset{\overline{\sigma_3}\cdot\exists ?l_1 \curvearrowright l_2\cdot\overline{\sigma_4}\cdot\mathtt{OK}}{\Longrightarrow}$. By induction, $\sigma_1 \cdot \sigma_3 \cdot l_1 \curvearrowright l_2 \cdot \sigma_4 \preceq \sigma''$; we can conclude by prefix closure and laws (L2) and (L3).

(v) $\phi \triangleq \exists ?(\nu\widetilde{l})\ \langle l \rangle @ l_1 : l_2$ **.** This is the most tedious case: $\amalg(\sigma)$ has a lot of possible evolutions and, thus, σ' can be of several forms. However, notice that the hypothesis of this Lemma reduces the number of such possibilities and the definition of canonical observers forces $\amalg(\sigma)$ to reduce to $(\nu\widetilde{l})(N \parallel \mathtt{test} :: P \parallel l_1 :: \langle l \rangle \parallel \{l_1 \leftrightarrow l_2\})$ in order to report success. For the sake of simplicity, we only consider the case for $\widetilde{l} = \emptyset$; the case for $\widetilde{l} = \{l\}$ is only notationally heavier. Let $H \triangleq l_1 :: \langle l \rangle \parallel \{l_1 \leftrightarrow l_2\}$; then, H can only exhibit six traces that, by using Proposition 5.2, can be combined in several ways with the traces of $\amalg(\sigma'') \triangleq N \parallel \mathtt{test} :: P$ to yield $\overline{\sigma'}$, as reported below (again, we ignore here labels of kind $l \curvearrowright l$, for $l \in \{l_1, l_2\}$).

1. ϵ *(this case corresponds to Proposition 5.2(1)).* Thus, $\amalg(\sigma'') \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$ and we can conclude by induction and by using (L1).
2. $l_1 \curvearrowright l_2$ (the case for $l_2 \curvearrowright l_1$ proceeds by symmetry). Similarly to case **(iv)**, we have three possibilities for $\overline{\sigma'}$. In all of them, the proof proceeds like the corresponding proof of case **(iv)**, by furtherly using law (L9) to rewrite $\exists ?\ \langle l \rangle @ l_1 : l_2$ into $\exists ?\ \langle l \rangle @ l_1 : l_1\ \cdot \exists ?l_1 \curvearrowright l_2$ and law (L1) to delete $\exists ?\ \langle l \rangle @ l_1 : l_1$.

3. $\langle l \rangle @ l_1 : l_1$. In this case, $\overline{\sigma'} = \overline{\sigma_1} \cdot \overline{\sigma_2}$, where $N \overset{\overline{\sigma_1}}{\Longrightarrow} N'$ and $N' \parallel \mathtt{test} :: P \parallel H \overset{\overline{\sigma_2} \cdot \mathtt{OK}}{\Longrightarrow}$. We then have three possibilities for $\overline{\sigma_2}$:

   (a) $\overline{\sigma_2} \triangleq \overline{\sigma_3} \cdot \langle l \rangle @ l_1 : l_1 \cdot \overline{\sigma_4}$ *and* $\text{Ш}(\sigma'') \overset{\overline{\sigma_3} \cdot \overline{\sigma_4} \cdot \mathtt{OK}}{\Longrightarrow}$ *(this case corresponds to the symmetric version of Proposition 5.2(1)).* The thesis follows by induction, prefi x closure and laws (L2), (L9) and (L1).

   (b) $\overline{\sigma_2} \triangleq \overline{\sigma_3} \cdot \langle l \rangle @ l_1 : l_3 \cdot \overline{\sigma_4}$ *and* $\text{Ш}(\sigma'') \overset{\overline{\sigma_3} \cdot l_1 \curvearrowright l_3 \cdot \overline{\sigma_4} \cdot \mathtt{OK}}{\Longrightarrow}$ *(this case corresponds to the symmetric version of Proposition 5.2(2)).* The thesis follows by induction, prefi x closure and laws (L2), (L9), (L1) and (L8).

   (c) $\overline{\sigma_2} \triangleq \overline{\sigma_3} \cdot \overline{\sigma_4}$ *and* $\text{Ш}(\sigma'') \overset{\overline{\sigma_3} \cdot \exists ? \langle l \rangle @ l_1 : l_1 \cdot \overline{\sigma_4} \cdot \mathtt{OK}}{\Longrightarrow}$ *(this case corresponds to Proposition 5.2(4)).* The thesis follows by induction, prefi x closure and laws (L9), (L1), (L2) and (L3).

4. $\langle l \rangle @ l_1 : l_2$. Again, $\overline{\sigma'} = \overline{\sigma_1} \cdot \overline{\sigma_2}$, where $N \overset{\overline{\sigma_1}}{\Longrightarrow} N'$ and $N' \parallel \mathtt{test} :: P \parallel H \overset{\overline{\sigma_2} \cdot \mathtt{OK}}{\Longrightarrow}$. Since the observer cannot perform remote inputs, the only possibility for $\overline{\sigma_2}$ is $\overline{\sigma_2} \triangleq \overline{\sigma_3} \cdot \langle l \rangle @ l_1 : l_2 \cdot \overline{\sigma_4}$, where $\text{Ш}(\sigma'') \overset{\overline{\sigma_3} \cdot \overline{\sigma_4} \cdot \mathtt{OK}}{\Longrightarrow}$. We then work like in case **(v)**.3(a), but without using laws (L9) and (L1).

5. $l_1 \curvearrowright l_2 \cdot \langle l \rangle @ l_1 : l_1$. Obtained by combining cases **(v)**.2 and **(v)**.3 above.

6. $\langle l \rangle @ l_1 : l_1 \cdot l_1 \curvearrowright l_2$. Obtained by combining cases **(v)**.3 and **(v)**.2 above. ∎

Finally, we can prove that trace equivalence exactly captures may testing.

Theorem 5.13 (Completeness of $\asymp$ w.r.t. $\simeq$) *If $N \simeq M$ then $N \asymp M$.*

Proof: Let $N \overset{\sigma}{\Longrightarrow}$; by Proposition 5.10, it holds that $N \parallel \text{Ш}(\sigma) \overset{\mathtt{OK}}{\Longrightarrow}$. By Proposition 3.9 and Defi nition 3.8, it holds that $M \parallel \text{Ш}(\sigma) \overset{\mathtt{OK}}{\Longrightarrow}$. By Lemma 5.11, there exists σ' such that $M \overset{\sigma'}{\Longrightarrow}$, $\text{Ш}(\sigma) \overset{\overline{\sigma'} \cdot \mathtt{OK}}{\Longrightarrow}$ and $\overline{\sigma'}$ does not contain labels of the form $\exists ? l$. Since $\mathtt{test}$ is fresh for N and M, it holds that $\mathtt{test} \notin n(\sigma') = n(\overline{\sigma'})$; thus, by Lemma 5.12, $\sigma' \preceq \sigma$, as required by Defi nition 5.4. ∎

From Theorems 5.8 and 5.13 we get the wanted result.

Corollary 5.14 (Alternative Characterization of May Testing) $\asymp = \simeq$.

6 Controlling the Activation of a Connection

TKLAIM can be easily accommodated to model a fi ner scenario where a handshake between the nodes involved is necessary to activate a connection (this feature is similar to the so-called *co-capabilities* of Safe Ambients [27]). This mechanism can be implemented by introducing a new action **acpt** that, by synchronizing with an action **conn**, authorises the activation of a new connection either from a specifi c node or from any node. An enabling action corresponding to **disc** seems to be less reasonable, as disconnections are usually unilateral (and, then, asynchronous) events. We extend the syntax of actions from Table 1 as follows:

$$a ::= \ \ldots \ \big| \ \mathbf{acpt}(!x) \ \big| \ \mathbf{acpt}(u)$$

Intuitively, action $\mathbf{acpt}(l)$ executed by a process located at l' means that l' is ready to activate a connection with l. Similarly, action $\mathbf{acpt}(!x)$ executed by a process located at l' means that l' is ready to activate a connection with any node, whose address will replace x in the continuation; thus, **acpt**$(!x).P$ binds x in P. These ideas are formalised by replacing in Table 4 rule (R-CONN) with

$$(\text{R-CONN}_1)\quad l_1 :: \mathbf{conn}(l_2).P \parallel l_2 :: \mathbf{acpt}(l_1).Q \;\longmapsto\; l_1 :: P \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: Q$$

$$(\text{R-CONN}_2)\quad l_1 :: \mathbf{conn}(l_2).P \parallel l_2 :: \mathbf{acpt}(!x).Q \;\longmapsto\; l_1 :: P \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: Q[^{l_1}/_x]$$

We believe that both the two forms of **acpt** are useful in practice. On the one hand, **acpt**$(!x)$ can be exploited by a server willing to accept connection requests from any, initially unknown, client. Notice that this form of client-server interaction could not be flexibly implemented by resorting to a shared TS storing connection requests, because a connection between the node hosting the TS and that of a potential client should be already in place for the client be able to put its request. On the other hand, **acpt**(l) should be used if a process is ready to activate connections only with a specifi c partner. Indeed, accepting connection requests from any process through **acpt**$(!x)$ and then, after checking the partner identity, disconnecting the unwanted partners through **disc**, could expose a node to security risks because the sequence of actions is not guaranteed to be performed atomically.

Defi nitions 3.4 and 3.5 are formally unchanged in this fi ner scenario; we still denote the resulting equivalences with symbols $\simeq$ and $\cong$. We now linger on their alternative characterisations: we start by developing a revised LTS that handles the new language; then, in Section 6.2 we present a sound and complete proof-technique for barbed congruence, while in Section 6.3 we touch upon a sound (but not complete) trace-based proof-technique for may testing.

6.1 A Revised Labelled Transition System

The LTS of Section 4.1 must be now extended with two new, complementary, labels: one for action **conn** and one for **acpt**. Thus, the syntax of labels becomes as follows:

$$\alpha \;::=\; \ldots \;\Big|\; (\nu\widetilde{l})l_1 : ?l_2 \;\Big|\; l_1 : !l_2$$

Intuitively, $(\nu\widetilde{l})l_1 : ?l_2$ results from enriching label $\exists ?\, l_2$ (that in the LTS of Section 4.1 is pointed out when action **conn**(l_2) is performed) with the node address l_1 (that can also be restricted) where the action is executed; of course, $bn(\,(\nu\widetilde{l})l_1 : ?l_2\,) \triangleq \widetilde{l}$. Label $l_1 : !l_2$ is instead pointed out when an action **acpt** accepting a connection request from l_2 is performed at l_1. These new labels are generated by rules (LTS-CONN), (LTS-ACC$_1$) and (LTS-ACC$_2$) in Table 9; they are synchronised via rule (LTS-EST), which activates a new connection as a consequence of a synchronization between a connection request and an acceptance. Like for (LTS-COMPL), no scope extrusion is carried out by (LTS-EST): the scope must have been extended previously through (LTS-STRUCT) (this also ensures the freshness of the node performing the **conn** for the net where the **acpt** is performed).

The new version of rule (LTS-OPEN) in Table 9 allows restricted nodes to perform action **conn**; however, it does not admit labels of the form $\langle l' \rangle @ l' : l$. Indeed, in the new framework, exporting a bound name via a communication does not 'fully open' its scope. Consider, for example, the net

$$(\nu l')(l :: \langle l' \rangle \parallel l' :: C)$$

It would be too informative to state that $(\nu l')(l :: \langle l' \rangle \parallel l' :: C) \xrightarrow{(\nu l')\,\langle l' \rangle @ l : l} l :: \mathbf{nil} \parallel l' :: C$. Indeed, if $C \triangleq \langle\rangle$, no context can observe the datum at l', because l' is "unreachable" (i.e., it is not connected with any other node of the net, nor it requires/accepts any connection). Hence, the nets

$$(\nu l')(l :: \langle l' \rangle \parallel l' :: \langle\rangle) \qquad \text{and} \qquad (\nu l')(l :: \langle l' \rangle \parallel l' :: \mathbf{nil})$$

(LTS-ACC₁)
$$l_1 :: \textbf{acpt}(l_2).P \xrightarrow{l_1 : !l_2} l_1 :: P$$

(LTS-ACC₂)
$$l_1 :: \textbf{acpt}(!x).P \xrightarrow{l_1 : !l_2} l_1 :: P[^{l_2}/_{x}]$$

(LTS-CONN)
$$l_1 :: \textbf{conn}(l_2).P \xrightarrow{l_1 : ?l_2} l_1 :: P$$

(LTS-EST)
$$\frac{\mathtt{N}_1 \xrightarrow{l_1 : ?l_2} \mathtt{N}_1' \qquad \mathtt{N}_2 \xrightarrow{l_2 : !l_1} \mathtt{N}_2'}{\mathtt{N}_1 \parallel \mathtt{N}_2 \xrightarrow{\tau} \mathtt{N}_1' \parallel \mathtt{N}_2' \parallel \{l_1 \leftrightarrow l_2\}}$$

(LTS-OPEN)
$$\frac{\mathtt{N} \xrightarrow{l : ?l'} \mathtt{N}' \qquad l' \neq l}{(\nu l)\mathtt{N} \xrightarrow{(\nu l)\ l : ?l'} \mathtt{N}'}$$

(LTS-HALFOPEN)
$$\frac{\mathtt{N} \xrightarrow{\langle l\rangle @ l_1 : l_2} \mathtt{N}' \qquad l \notin \{l_1, l_2\}}{(\nu l)\mathtt{N} \xrightarrow{(\nu l)\ \langle l\rangle @ l_1 : l_2} (l)\mathtt{N}'}$$

(LTS-FULLOPEN)
$$\frac{\mathtt{N} \xrightarrow{\alpha} \mathtt{N}' \qquad \alpha \in \{l \curvearrowright l',\ l : ?l'\ ,\ l : !l'\ \} \qquad l' \neq l}{(l)\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'}$$

(LTS-HALFRES₁)
$$\frac{\mathtt{N} \xrightarrow{\alpha} \mathtt{N}' \qquad l \notin n(\alpha)}{(l)\mathtt{N} \xrightarrow{\alpha} (l)\mathtt{N}'}$$

(LTS-HALFRES₂)
$$\frac{\mathtt{N} \xrightarrow{\alpha} \mathtt{N}' \qquad \alpha \in \{\ \langle l\rangle @ l_1\ :\ l_2\ , \exists ?\ \langle l\rangle @ l_1\ :\ l_2\ \} \qquad l \notin \{l_1, l_2\}}{(l)\mathtt{N} \xrightarrow{\alpha} (l)\mathtt{N}'}$$

plus all rules from Table 5, but (LTS-CONN) and (LTS-OPEN), with $\mathtt{N}$ in place of N everywhere and with the extended $\equiv$.

Table 9: The Revised LTS

are equated by both $\cong$ and $\simeq$. Fully opening the scope of l' would give to the bisimilarity and the trace equivalence an observational power that no context of the language has.

To properly tackle these situations, we say that the scope of an extruded name is only *half-opened* by a label of the form $(\nu l')\ \langle l'\rangle @ l\ :\ l$ and introduce the notion of *extended nets*, that are nets possibly containing *half-restricted* names. Extended nets are ranged over by $\mathtt{N}, \mathtt{M}, \mathtt{K}, ...$ (and their decorated versions) and are formally defined as follows:

$$\mathtt{N}\ ::=\ N\ \big|\ (l)\mathtt{N}\ \big|\ (\nu l)\mathtt{N}\ \big|\ \mathtt{N}_1 \parallel \mathtt{N}_2$$

Intuitively, half-restricted names correspond to addresses of nodes whose scope has been extended but whose reachability is still unknown. Thus, the half-restriction operator (l) is not a binder for l, because l is known to the environment (that has previously received l via an action $(\nu l)\ \langle l\rangle @ l_1\ :\ l_2$); so, $fn((l)\mathtt{N}) \triangleq \{l\} \cup fn(\mathtt{N})$ and, in $(l)\mathtt{N}$, l cannot be alpha-converted. The following rules extend structural equivalence to cope with half-restrictions too:

(HCOM)	$(l_1)(l_2)\mathtt{N} \equiv (l_2)(l_1)\mathtt{N}$	(RHCOM)	$(l_1)(\nu l_2)\mathtt{N} \equiv (\nu l_2)(l_1)\mathtt{N}$
(HGARB)	$(l)\mathbf{0} \equiv \mathbf{0}$	(HEXT)	$\mathtt{N} \parallel (l)\mathtt{M} \equiv (l)(\mathtt{N} \parallel \mathtt{M})$ if $l \notin fa(\mathtt{N})$

where we write $l \in fa(N)$ if $N \equiv N \parallel l :: \textbf{nil}$, i.e. l is the address of a non-restricted node in N (we shall sometimes say that l is a *free address* of N, and this motivates the notation). The last rule is justified by the fact that, since l is half-restricted, it has been previously exported via a bound output; thus, it can occur as a free name in $\mathtt{N}$ (maybe, in the receiving process), but it cannot be a free address of $\mathtt{N}$, since it is still (potentially) unreachable in $\mathtt{M}$.

Thus, a label $(\nu l)\ \langle l\rangle @ l_1 : l_2$ is pointed out when a restriction on l is turned to a half-restriction, see rule (LTS-HALFOPEN) in Table 9. Half-restrictions are removed only when the node corresponding to a half-restricted name becomes 'reachable', i.e. whenever it performs a **conn**/**acpt** or whenever it exhibits a connection with another node of the net (see rule (LTS-FULLOPEN)). Until such a moment, actions involving a half-restricted name l are regulated by rules (LTS-HALFRES$_1$), that permits all those actions not involving l, and (LTS-HALFRES$_2$), that permits output and input of l. This should motivate the term *half-restriction*: a half-restricted name is not either really restricted nor free, since only some actions involving it are forbidden.

Summarizing, the revised labelled transition relation (between extended nets) is presented in Table 9. Notably, its τ-steps coincide with the reductions of the extended language (that is, an analogous of Proposition 4.4 still holds). Moreover, an analogous of Proposition 4.2 holds; in what follows, whenever we mention Proposition 4.2, we intend its analogous for the LTS of Table 9. Finally, Proposition 4.3 becomes as follows (in particular, cases 2., 3. and 6. are similar to the corresponding cases); the proof of the following result is omitted since it is very similar to the proof of Proposition 4.3.

Proposition 6.1 *Let $\widetilde{l_1} \cap \widetilde{l_2} = \emptyset$; then, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N} \parallel \mathtt{K}) \xrightarrow{\alpha} \overline{\mathtt{N}}$ if and only if one of the following conditions holds:*

1. $(\nu\widetilde{l_1})(\widetilde{l_2})\mathtt{N} \xrightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})\mathtt{N}'$ *and* $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{N}' \parallel \mathtt{K})$. *In particular*

   (a) $n(\alpha) \cap \{\widetilde{l_1}, \widetilde{l_2}\} = \emptyset$ *implies that* $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$, $\widetilde{l'_1} = \widetilde{l_1}$ *and* $\widetilde{l'_2} = \widetilde{l_2}$

   (b) $\alpha = (\nu l)\alpha'$, $\alpha' = \langle l\rangle @ l_1 : l_2$, $l \in \widetilde{l_1}$ *and* $\{l_1, l_2\} \cap \{\widetilde{l_1}, \widetilde{l_2}\} = \emptyset$ *imply that* $\mathtt{N} \xrightarrow{\alpha'} \mathtt{N}'$, $\widetilde{l'_1} = \widetilde{l_1} - \{l\}$ *and* $\widetilde{l'_2} = \widetilde{l_2} \cup \{l\}$

   (c) $\alpha \in \{\langle l\rangle @ l_1 : l_2\ ,\ \exists ?\ \langle l\rangle @ l_1 : l_2\}$, $l \in \widetilde{l_2}$ *and* $\{l_1, l_2\} \cap \{\widetilde{l_1}, \widetilde{l_2}\} = \emptyset$ *imply that* $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$, $\widetilde{l'_1} = \widetilde{l_1}$ *and* $\widetilde{l'_2} = \widetilde{l_2}$

   (d) $\alpha = (\nu l)\alpha'$, $\alpha' = l : ?l'$, $l \in \widetilde{l_1}$ *and* $l' \notin \{\widetilde{l_1}, \widetilde{l_2}\}$ *imply that* $\mathtt{N} \xrightarrow{\alpha'} \mathtt{N}'$, $\widetilde{l'_1} = \widetilde{l_1} - \{l\}$ *and* $\widetilde{l'_2} = \widetilde{l_2}$

   (e) $\alpha \in \{l \curvearrowright l',\ l : ?l'\ ,\ l : !l'\ \}$, $l \in \widetilde{l_2}$ *and* $l' \notin \{\widetilde{l_1}, \widetilde{l_2}\}$ *imply that* $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$, $\widetilde{l'_1} = \widetilde{l_1}$ *and* $\widetilde{l'_2} = \widetilde{l_2} - \{l\}$

2. $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_1 : l_1} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{K}'$ *and* $\overline{\mathtt{N}} \equiv (\nu\widetilde{l''})(\mathtt{N}' \parallel \mathtt{K}')$; *moreover, if* $\widetilde{l'} = \emptyset$ *and* $l \in \widetilde{l}$, *then* $\alpha = (\nu l)\ \langle l\rangle @ l_1 : l_2$ *and* $\widetilde{l''} = \widetilde{l} - \{l\}$; *otherwise,* $\alpha = (\nu\widetilde{l'})\ \langle l\rangle @ l_1 : l_2$ *and* $\widetilde{l''} = \widetilde{l}$

3. $\mathtt{N} \xrightarrow{\exists ? l_1 \curvearrowright l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{K}'$, $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$ *and* $\alpha = \tau$

4. *(a)* $\mathtt{N} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} (\widetilde{l})\mathtt{N}'$, $\mathtt{K} \xrightarrow{(\nu\widetilde{l})\ \langle l\rangle @ l_2 : l_1} \mathtt{K}'$, $\widetilde{l} \cap fn(\mathtt{N}) = \emptyset$, $\overline{\mathtt{N}} \equiv (\nu\widetilde{l}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$ *and* $\alpha = \tau$

   (b) $\mathtt{N} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{N}'$, $\mathtt{K} \equiv (l)\mathtt{K}'$, $\mathtt{K}' \xrightarrow{\langle l\rangle @ l_2 : l_1} \mathtt{K}''$, $l \notin fa(\mathtt{N})$, $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(l, \widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}'')$ *and* $\alpha = \tau$

5. *(a)* $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{N}'$, $\mathtt{K} \xrightarrow{(\nu\widetilde{l})\ \langle l\rangle @ l_2 : l_2} \mathtt{K}'$, $\widetilde{l} \cap fn(\mathtt{N}) = \emptyset$, $\overline{\mathtt{N}} \equiv (\nu\widetilde{l}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$ *and* $\alpha = \tau$

   (b) $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{N}'$, $\mathtt{K} \equiv (l)\mathtt{K}'$, $\mathtt{K}' \xrightarrow{\langle l\rangle @ l_2 : l_2} \mathtt{K}''$, $l \notin fa(\mathtt{N})$, $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(l, \widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}'')$ *and* $\alpha = \tau$

6. $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\,\langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\,\langle l\rangle @ l_2 : l_1} \mathtt{N}', \mathtt{K} \xrightarrow{l_2 \curvearrowright l_1} \mathtt{K}', \widetilde{l'} \cap fn(\mathtt{K}) = \emptyset, \overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$ *and* $\alpha = \tau$

7. $\mathtt{N} \xrightarrow{(\nu\widetilde{l})l_1 : ?l_2} \mathtt{N}', \mathtt{K} \xrightarrow{l_2 : !l_1} \mathtt{K}', \widetilde{l} \cap fn(\mathtt{K}) = \emptyset, \overline{\mathtt{N}} \equiv (\nu\widetilde{l_1}, \widetilde{l})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}' \parallel \{l_1 \leftrightarrow l_2\})$ *and* $\alpha = \tau$

8. *one of the previous cases with K in place of N and vice versa.*

Finally, we give a simple Proposition that describes how the free addresses of a net change upon execution of a transition; the proof straightforwardly follows from the definition of the LTS. In particular, notice that half-restricted names are not free addresses, because of the side condition of rule (HEXT).

Proposition 6.2 *Let* $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$*; then*

- *if* $\alpha \in \{\, \tau, (\nu\widetilde{l})\,\langle l\rangle @ l_1 : l_2 \,\}$ *then* $fa(\mathtt{N}') = fa(\mathtt{N})$
- *if* $\alpha \in \{\, l \curvearrowright l', (\nu\widetilde{l})\, l : ?l',\ l : !l' \,\}$ *then* $fa(\mathtt{N}') = fa(\mathtt{N}) \cup \{l\}$*, if l is half-restricted in* N*, and* $fa(\mathtt{N}') = fa(\mathtt{N}) \cup \widetilde{l}$*, otherwise*
- *if* $\alpha \in \{\, \exists ?l_1 \curvearrowright l_2, \exists ?\,\langle l\rangle @ l_2 : l_1 \,\}$ *then* $fa(\mathtt{N}') = fa(\mathtt{N}) \cup \{l_2\}$.

Proof: The proof can be done by an easy induction on the depth of the shortest inference of the judgement $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$. ∎

6.2 A Bisimulation-based Characterisation of Barbed Congruence

We can softly adapt Definition 4.5 to characterise barbed congruence also in the richer language. We only remark that actions **conn** and **acpt** correspond quite closely to the output/input prefixes of the synchronous π-calculus [29]. Thus, they are handled in a 'traditional' way, see point 1. of Definition 6.3 below.

Definition 6.3 (Bisimilarity) *A symmetric relation* $\Re$ *between* TKLAIM *nets is a* bisimulation *if, for each* $N \Re M$ *and* $N \xrightarrow{\alpha} N'$*, it holds that:*

1. $\alpha \in \{\tau, l_1 \curvearrowright l_2, (\nu\widetilde{l})\,\langle l\rangle @ l_1 : l_1\,,\ (\nu\widetilde{l})l_1 : ?l_2\,,\ l_1 : !l_2 \,\}$ *implies that* $M \overset{\hat{\alpha}}{\Longrightarrow} M'$ *and* $N' \Re M'$*, for some* M'*;*
2. $\alpha = \exists ?\beta$ *implies that* $M \parallel \text{NET}(\beta) \Rightarrow M'$ *and* $N' \Re M'$*, for some* M'.

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

We now prove that this new version of the bisimilarity still exactly captures barbed congruence. We follow the path of Section 4.2 to prove the main result of this section, as reported by the following theorem.

Theorem 6.4 (Alternative Characterisation of Barbed Congruence) $\approx\ =\ \cong$.

The inclusion "$\subseteq$" trivially follows from the fact that $\approx$ is context closed. The inclusion "$\supseteq$" follows from the fact that $\cong$ is a bisimulation; thanks to context closure, this can be proved by building, for any possible action, a context forcing two barbed congruent nets to behave as required by Definition 6.3. In the remainder of this section, we give full details on these key steps.

Lemma 6.5 $\approx$ *is context closed.*

Proof: We shall prove that the relation

$$\Re \triangleq \{(\ (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N} \parallel \mathtt{K})\ ,\ (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K})\ )\ :\ (\widetilde{l})\mathtt{N} \approx (\widetilde{l})\mathtt{M}, \widetilde{l} \subseteq (\widetilde{l_1}, \widetilde{l_2}), \widetilde{l} \cap fa(\mathtt{K}) = \emptyset,$$
$$K \text{ is restriction and half-restriction free } \}$$

is a bisimulation up-to $\equiv$; by taking $\widetilde{l} = \emptyset$ we obtain the thesis. Consider $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N} \parallel \mathtt{K}) \xrightarrow{\alpha} \overline{\mathtt{N}}$; by Proposition 6.1 we have the following cases (notice that, since $\mathtt{K}$ is restriction and half-restriction free, cases 4(b) and 5(b) do not occur; moreover, there is no restriction on the label from $\mathtt{K}$ in cases 4(a) and 5(a) and in the symmetric of cases 2 and 7):

1. *(Proposition 6.1(1))* $(\nu\widetilde{l_1})(\widetilde{l_2})\mathtt{N} \xrightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})\mathtt{N}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{N}' \parallel \mathtt{K})$; we have five sub-cases:

   (a) $(\widetilde{l_1}, \widetilde{l_2}) \cap n(\alpha) = \emptyset$**,** $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$**,** $\widetilde{l'_1} = \widetilde{l_1}$ **and** $\widetilde{l'_2} = \widetilde{l_2}$**:** then, $(\widetilde{l})\mathtt{N} \xrightarrow{\alpha} (\widetilde{l})\mathtt{N}'$; we reason by case analysis on α:

      i. $\alpha \in \{\tau,\ l_1 \curvearrowright l_2, (\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_1\ ,\ (\nu\widetilde{l})l_1 : ?l_2\ ,\ l_1 : !l_2\ \}$. By hypothesis, $(\widetilde{l})\mathtt{M} \overset{\hat{\alpha}}{\Longrightarrow} (\widetilde{l})\mathtt{M}'$ and $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$; thus, trivially, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \overset{\hat{\alpha}}{\Longrightarrow} (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \overline{\mathtt{M}}$ and, by definition, $\overline{\mathtt{N}}\ \Re\ \overline{\mathtt{M}}$.

      ii. $\alpha = \exists ?l_1 \curvearrowright l_2$. Then, since $(\widetilde{l_1}, \widetilde{l_2}) \cap \{l_1, l_2\} = \emptyset$, we have $(\widetilde{l})\mathtt{M} \parallel \{l_1 \leftrightarrow l_2\} \equiv (\widetilde{l})(\mathtt{M} \parallel \{l_1 \leftrightarrow l_2\}) \Rightarrow (\widetilde{l})\mathtt{M}'$ and $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$; hence, it holds that $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \parallel \{l_1 \leftrightarrow l_2\} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \{l_1 \leftrightarrow l_2\} \parallel \mathtt{K}) \Rightarrow (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \overline{\mathtt{M}}$ and $\overline{\mathtt{N}}\ \Re\ \overline{\mathtt{M}}$.

      iii. $\alpha = \exists ?\ \langle l\rangle @ l_2 : l_1$. This is similar to the previous case: by (EXT) and (HEXT), $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \parallel \mathtt{K})$.

   (b) $\alpha = (\nu l)\alpha'$**,** $\alpha' = \ \langle l\rangle @ l_1 : l_2\ $**,** $l \in \widetilde{l_1}$**,** $\{l_1, l_2\} \cap (\widetilde{l_1}, \widetilde{l_2}) = \emptyset$**,** $\mathtt{N} \xrightarrow{\alpha'} \mathtt{N}'$**,** $\widetilde{l'_1} = \widetilde{l_1} - \{l\}$ **and** $\widetilde{l'_2} = \widetilde{l_2} \cup \{l\}$**:** then, it can be either $l \in \widetilde{l}$ or not; however, in both cases, $(\widetilde{l})\mathtt{M} \overset{\alpha'}{\Longrightarrow} (\widetilde{l})\mathtt{M}'$ and $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$. Then, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \overset{\alpha}{\Longrightarrow} (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \overline{\mathtt{M}}$ and $\overline{\mathtt{N}}\ \Re\ \overline{\mathtt{M}}$, since $\widetilde{l}$ is still a subset of $(\widetilde{l'_1}, \widetilde{l'_2})$.

   (c) $\alpha \in \{\ \langle l\rangle @ l_1 : l_2\ , \exists ?\ \langle l\rangle @ l_1 : l_2\ \}$**,** $l \in \widetilde{l_2}$**,** $\{l_1, l_2\} \cap (\widetilde{l_1}, \widetilde{l_2}) = \emptyset$**,** $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$**,** $\widetilde{l'_1} = \widetilde{l_1}$ **and** $\widetilde{l'_2} = \widetilde{l_2}$**:** the case for $\alpha = \ \langle l\rangle @ l_1 : l_2\ $ is trivial, whereas the case for $\alpha = \exists ?\ \langle l\rangle @ l_1 : l_2$ is similar to case 1(a).iii.

   (d) $\alpha = (\nu l)\alpha'$**,** $\alpha' = \ l : ?l'\ $**,** $l \in \widetilde{l_1}$**,** $l' \notin (\widetilde{l_1}, \widetilde{l_2})$**,** $\mathtt{N} \xrightarrow{\alpha'} \mathtt{N}'$**,** $\widetilde{l'_1} = \widetilde{l_1} - \{l\}$ **and** $\widetilde{l'_2} = \widetilde{l_2}$**:** if $l \notin \widetilde{l}$ then the case is simple. Otherwise, $(\widetilde{l})\mathtt{N} \overset{\alpha'}{\Longrightarrow} (\widetilde{l'})\mathtt{N}'$, for $\widetilde{l'} = \widetilde{l} - \{l\}$; thus, $(\widetilde{l})\mathtt{M} \overset{\alpha'}{\Longrightarrow} (\widetilde{l'})\mathtt{M}'$ and $(\widetilde{l'})\mathtt{N}' \approx (\widetilde{l'})\mathtt{M}'$. Then, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \overset{\alpha}{\Longrightarrow} (\nu\widetilde{l'_1})(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \overline{\mathtt{M}}$ and $\overline{\mathtt{N}}\ \Re\ \overline{\mathtt{M}}$, since $\widetilde{l'} \subseteq \widetilde{l}$ and, hence, $\widetilde{l'} \cap fa(\mathtt{K}) = \emptyset$.

   (e) $\alpha \in \{l \curvearrowright l',\ l : ?l'\ ,\ l : !l'\ \}$**,** $l \in \widetilde{l_2}$**,** $l' \notin (\widetilde{l_1}, \widetilde{l_2})$**,** $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$**,** $\widetilde{l'_1} = \widetilde{l_1}$ **and** $\widetilde{l'_2} = \widetilde{l_2} - \{l\}$**:** similar to case 1(d).

2. *(symmetric of Proposition 6.1(1))* $(\nu\widetilde{l_1})(\widetilde{l_2})\mathtt{K} \xrightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})\mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{N} \parallel \mathtt{K}')$; since we are working up-to $\equiv$, by using laws (EXT) and (HEXT), we can assume that $\mathtt{K}'$ is restriction and half-restriction free. Points (b)/.../(e) are similar to the corresponding cases in point 1; for case (a), we reason by case analysis on α:

   (i) $\alpha = \tau$. Then, trivially, $\widetilde{l_1} \subseteq \widetilde{l'_1}$ ('$\subset$' holds whenever $\mathtt{K}$ evolves by performing a **new**) and $\widetilde{l'_2} = \widetilde{l_2}$. Thus, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \xrightarrow{\tau} (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{M} \parallel \mathtt{K}') \triangleq \overline{\mathtt{M}}$; by definition, $\overline{\mathtt{N}}\ \Re\ \overline{\mathtt{M}}$, since $\widetilde{l} \subseteq (\widetilde{l'_1}, \widetilde{l'_2})$ and $fa(\mathtt{K}') = fa(\mathtt{K})$ (see Proposition 6.2).

(ii) $\alpha = \langle l\rangle @ l_1 : l_2$. By Proposition 6.2, $fa(\mathtt{K}') = fa(\mathtt{K})$. Moreover, by definition of the LTS, it must be $\{l_1, l_2\} \cap (\widetilde{l_1}, \widetilde{l_2}) = \emptyset$; hence, we have three subcases:

- $l \notin (\widetilde{l_1}, \widetilde{l_2})$: similar case 2(i), with $\widetilde{l'_1} = \widetilde{l_1}$.
- $l \in \widetilde{l_1}$: then, $\widetilde{l'_1} \triangleq \widetilde{l_1} - \{l\}$ and $\widetilde{l'_2} = \widetilde{l_2} \cup \{l\}$. Since $\widetilde{l}$ is still a subset of $(\widetilde{l'_1}, \widetilde{l'_2})$, we conclude as before.
- $l \in \widetilde{l_2}$: then, $\widetilde{l'_1} = \widetilde{l_1}$ and $\widetilde{l'_2} = \widetilde{l_2}$; thus, we easily conclude.

(iii) $\alpha \in \{l_1 \frown l_2,\ l_1 : ?l_2,\ l_1 : !l_2\}$. Then, by definition of the LTS, $l_2 \notin (\widetilde{l_1}, \widetilde{l_2})$ and, because $l_1 \in fa(\mathtt{K})$, $l_1 \notin \widetilde{l}$. Moreover, if $l_1 \notin (\widetilde{l_1}, \widetilde{l_2})$, then $\widetilde{l'_1} = \widetilde{l_1}$ and $\widetilde{l'_2} = \widetilde{l_2}$; if $l_1 \in \widetilde{l_1}$, then $\widetilde{l'_1} = \widetilde{l_1} - \{l_1\}$ and $\widetilde{l'_2} = \widetilde{l_2}$; finally, if $l_1 \in \widetilde{l_2}$, then $\widetilde{l'_1} = \widetilde{l_1}$ and $\widetilde{l'_2} = \widetilde{l_2} - \{l_1\}$. Hence, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \xrightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{M} \parallel \mathtt{K}') \triangleq \overline{\mathtt{M}}$ and, by definition, $\overline{\mathtt{N}}\ \mathfrak{R}\ \overline{\mathtt{M}}$, since $\widetilde{l}$ is still a subset of $(\widetilde{l'_1}, \widetilde{l'_2})$ and $\widetilde{l} \cap fa(\mathtt{K}') = \emptyset$.

(iv) $\alpha = \exists ?l_1 \frown l_2$. Then $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \parallel \{l_1 \leftrightarrow l_2\} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K} \parallel \{l_1 \leftrightarrow l_2\}) \xrightarrow{\tau} (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}') \triangleq \overline{\mathtt{M}}$ and, trivially, $\overline{\mathtt{N}}\ \mathfrak{R}\ \overline{\mathtt{M}}$. Indeed, by Proposition 6.2, $fa(\mathtt{K}') = fa(\mathtt{K}) \cup \{l_2\}$ but, by definition of the LTS, $l_2 \notin (\widetilde{l_1}, \widetilde{l_2})$; thus, $fa(\mathtt{K}') \cap \widetilde{l} = \emptyset$.

(v) $\alpha = \exists ?\ \langle l\rangle @ l_2 : l_1$. This is similar to the previous case: indeed, it can at most be $l \in \widetilde{l_2}$. In that case, $l \notin \{l_1, l_2\}$ and, by (HEXT), we can easily conclude.

3. *(Proposition 6.1(2))* $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} (\widetilde{l'})\mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 \frown l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; this case is easy derivable from case 11.

4. *(symmetric of Proposition 6.1(2))* $\mathtt{N} \xrightarrow{l_1 \frown l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{\langle l\rangle @ l_2 : l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; this case is easy derivable from case 12.

5. *(Proposition 6.1(3))* $\mathtt{N} \xrightarrow{\exists ?l_1 \frown l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 \frown l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; then, since $\{l_1, l_2\} \subseteq fa(\mathtt{K})$, we have that $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$. Hence, $(\widetilde{l})\mathtt{M} \parallel \{l_1 \leftrightarrow l_2\} \equiv (\widetilde{l})(\mathtt{M} \parallel \{l_1 \leftrightarrow l_2\}) \Rightarrow (\widetilde{l})\mathtt{M}'$ and $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$. Now, by Proposition 4.2(1), $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \{l_1 \leftrightarrow l_2\} \parallel \mathtt{K}')$ and we can easily conclude.

6. *(symmetric of Proposition 6.1(3))* $\mathtt{N} \xrightarrow{l_1 \frown l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{\exists ?l_1 \frown l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; then, since $l_1 \in fa(\mathtt{K})$, we have that $l_1 \notin \widetilde{l}$. If $l_2 \notin \widetilde{l}$, then $(\widetilde{l})\mathtt{M} \xRightarrow{l_1 \frown l_2} (\widetilde{l})\mathtt{M}'$ and $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$; we can easily conclude (notice that $fa(\mathtt{K}') = fa(\mathtt{K}) \cup \{l_2\}$ and, hence, $fa(\mathtt{K}') \cap \widetilde{l} = \emptyset$). If $l_2 \in \widetilde{l}$, then we consider $\mathtt{N} \xrightarrow{l_2 \frown l_1} \mathtt{N}'$ (that must hold whenever $\mathtt{N} \xrightarrow{l_1 \frown l_2} \mathtt{N}'$ holds). Thus, $(\widetilde{l})\mathtt{N} \xrightarrow{l_2 \frown l_1} (\widetilde{l'})\mathtt{N}'$, for $\widetilde{l'} = \widetilde{l} - \{l_2\}$; then, $(\widetilde{l})\mathtt{M} \xRightarrow{l_2 \frown l_1} (\widetilde{l'})\mathtt{M}'$ and $(\widetilde{l'})\mathtt{N}' \approx (\widetilde{l'})\mathtt{M}'$; this suffices to conclude, as $fa(\mathtt{K}') \cap \widetilde{l'} = \emptyset$.

7. *(Proposition 6.1(4).a)* $\mathtt{N} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{N}'$, $\mathtt{K} \xrightarrow{\langle l\rangle @ l_2 : l_1} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; this case is similar to case 5, but also uses (HEXT).

8. *(symmetric of Proposition 6.1(4))* $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_1} \mathtt{N}'$ and $\mathtt{K} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{K}'$; then, we have two sub-cases:

   (a) $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_1} (\widetilde{l'})\mathtt{N}'$**,** $\widetilde{l'} \cap fn(\mathtt{K}) = \emptyset$ **and** $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$**:** again, by definition of the LTS, $l_1 \in fa(\mathtt{K})$. So, $l_1 \notin \widetilde{l}$ while it may be either $l_2 \in \widetilde{l}$ or not; we only consider the first case, as the second one is simpler. Then, $(\widetilde{l})\mathtt{N} \xrightarrow{l_2 \frown l_1} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} (\widetilde{l''})\mathtt{N}'$, where

$\widetilde{l''} = (\widetilde{l} - \{l_2\}) \cup \widetilde{l'}$. Thus, $(\widetilde{l})\mathtt{M} \xRightarrow{l_2 \curvearrowright l_1} \xRightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} (\widetilde{l''})\mathtt{M}'$ and $(\widetilde{l''})\mathtt{N}'' \approx (\widetilde{l''})\mathtt{M}'$. By Proposition 4.2, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \Rightarrow (\nu\widetilde{l'}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \parallel \mathtt{K}) \xrightarrow{\tau} (\nu\widetilde{l'}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}') \triangleq \overline{\mathtt{M}}$ and $\overline{\mathtt{N}}\ \mathfrak{R}\ \overline{\mathtt{M}}$: indeed, $\widetilde{l''} \subseteq (\widetilde{l'}, \widetilde{l_1}, \widetilde{l_2})$ and, because $\widetilde{l'} \cap fn(\mathtt{K}) = \emptyset$, it holds that $\widetilde{l''} \cap fa(\mathtt{K}') = \emptyset$.

(b) $\mathtt{N} \equiv (l)\mathtt{N}'$, $\mathtt{N}' \xrightarrow{\langle l\rangle @ l_2 : l_1} \mathtt{N}''$, $l \notin fa(\mathtt{K})$ **and** $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(l, \widetilde{l_2})(\mathtt{N}'' \parallel \mathtt{K}')$**:** this case can be proved like case 8(a); notice that here we have $\widetilde{l''} = (\widetilde{l} - \{l_2\})$.

9. *(Proposition 6.1(5).a)* $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{N}' \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{N}''$, $\mathtt{K} \xrightarrow{\langle l\rangle @ l_2 : l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}'' \parallel \mathtt{K}')$; by definition of $\mathtt{K}$ and of the LTS, it holds that $l_2 \notin \widetilde{l}$. On the other hand, it may be either $l_1 \in \widetilde{l}$ or not; we only explicitly consider the first case, that is more delicate. We now have $(\widetilde{l})\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} (\widetilde{l'})\mathtt{N}'$, with $\widetilde{l'} = \widetilde{l} - \{l\}$; then, $(\widetilde{l})\mathtt{M} \xRightarrow{l_1 \curvearrowright l_2} (\widetilde{l'})\mathtt{M}'$ and $(\widetilde{l'})\mathtt{N}' \approx (\widetilde{l'})\mathtt{M}'$. Now, $(\widetilde{l'})\mathtt{N}' \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} (\widetilde{l'})\mathtt{N}''$; thus, $(\widetilde{l'})\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \equiv (\widetilde{l'})(\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle) \Rightarrow (\widetilde{l'})\mathtt{M}''$ and $(\widetilde{l'})\mathtt{N}'' \approx (\widetilde{l'})\mathtt{M}''$. By Proposition 4.2(1) and 4.2(2), $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \Rightarrow (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \parallel \mathtt{K}') \Rightarrow (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}'' \parallel \mathtt{K}') \triangleq \overline{\mathtt{M}}$ and $\overline{\mathtt{N}}\ \mathfrak{R}\ \overline{\mathtt{M}}$.

10. *(symmetric of Proposition 6.1(5))*

    (a) $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} (\widetilde{l'})\mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 \curvearrowright l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{K}'$ **and** $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$**:** this is similar to case 6 but simpler, since $\widetilde{l} \cap \{l_1, l_2\} = \emptyset$.

    (b) $\mathtt{N} \equiv (l)\mathtt{N}' \xrightarrow{\langle l\rangle @ l_2 : l_2} (l)\mathtt{N}''$, $\mathtt{K} \xrightarrow{l_1 \curvearrowright l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{K}'$ **and** $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(l, \widetilde{l_2})(\mathtt{N}'' \parallel \mathtt{K}')$**:** this case is similar to case 8(a) but simpler, since $\widetilde{l} \cap \{l_1, l_2\} = \emptyset$.

11. *(Proposition 6.1(6))* $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} (\widetilde{l'})\mathtt{N}' \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} (\widetilde{l'})\mathtt{N}''$, $\mathtt{K} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{N}'' \parallel \mathtt{K}')$; by definition of $\mathtt{K}$, it holds that $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$. Hence, $(\widetilde{l})\mathtt{M} \xRightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} (\widetilde{l}, \widetilde{l'})\mathtt{M}'$, $(\widetilde{l}, \widetilde{l'})\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \equiv (\widetilde{l}, \widetilde{l'})(\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle) \Rightarrow (\widetilde{l}, \widetilde{l'})\mathtt{M}''$ and $(\widetilde{l}, \widetilde{l'})\mathtt{N}'' \approx (\widetilde{l}, \widetilde{l'})\mathtt{M}''$; we easily conclude up-to $\equiv$, since $(\widetilde{l}, \widetilde{l'}) \subseteq (\widetilde{l'}, \widetilde{l_1}, \widetilde{l_2})$ and $(\widetilde{l}, \widetilde{l'}) \cap fa(\mathtt{K}') = \emptyset$, because bound names are different from the free ones.

12. *(symmetric of Proposition 6.1(6))* $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{\langle l\rangle @ l_2 : l_2} \xrightarrow{\exists ?\ \langle l\rangle @ l_2 : l_1} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; notice that $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$ and easily conclude.

13. *(Proposition 6.1(7))* $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ l_2 : ?l_1} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 : !l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l'}, \widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}' \parallel \{l_1 \leftrightarrow l_2\})$; by definition of the LTS and by $\widetilde{l} \cap fa(\mathtt{K}) = \emptyset$, it holds that $l_1 \notin \widetilde{l}$. If $\widetilde{l'} = \{l_2\}$, then $(\widetilde{l})\mathtt{M} \xRightarrow{(\nu l_2)\ l_2 : ?l_1} (\widetilde{l})\mathtt{M}'$, for $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$, and we easily conclude. If $\widetilde{l'} = \emptyset$, we reason like in case 6.

14. *(symmetric of Proposition 6.1(7))* $\mathtt{N} \xrightarrow{l_2 : !l_1} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 : ?l_2} \mathtt{K}'$ and $\overline{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}' \parallel \{l_1 \leftrightarrow l_2\})$; similar to case 13. ∎

By proceeding as for Theorem 4.8, we can prove the following result.

Corollary 6.6 (Soundness) $\approx\ \subseteq\ \cong$.

We now consider the completeness part that can be easily proved, as before, once we prove the following Lemma (that generalises Lemma 4.9).

Lemma 6.7

1. *Let* $(\nu l)(N \parallel l_f :: \langle l \rangle) \cong (\nu l)(M \parallel l_f :: \langle l \rangle)$ *and* l_f *be fresh for* N*,* M *and* l*; then,* $(l)N \approx (l)M$*.*

2. *Let* $(\nu l)(N \parallel \{l \leftrightarrow l_f\} \parallel l_f :: \langle l \rangle) \cong (\nu l)(M \parallel \{l \leftrightarrow l_f\} \parallel l_f :: \langle l \rangle)$*,* l_f *be fresh for* N*,* M *and* l*, and* $l \in fa(N) \cap fa(M)$*; then,* $N \approx M$*.*

Proof: We shall prove the two claims at once. To this aim, let $\widetilde{l_1} \triangleq \{l_1, \ldots, l_k\}$ and $\widetilde{l_2} \triangleq \{l'_1, \ldots, l'_h\}$ such that $h, k \geq 0$ and $\widetilde{l_1} \cap \widetilde{l_2} = \emptyset$. Let $\widetilde{f_1} \triangleq \{f_1, \ldots, f_k\}$ and $\widetilde{f_2} \triangleq \{f'_1, \ldots, f'_h\}$ be distinct, fresh and reserved names. Finally, let

$$[\![N, \widetilde{l_1}, \widetilde{f_1}, \widetilde{l_2}, \widetilde{f_2}]\!] \triangleq (\nu \widetilde{l_1}, \widetilde{l_2})(N \parallel \prod_{i=1}^{k} f_i :: \langle l_i \rangle \parallel \prod_{j=1}^{h} (f'_j :: \langle l'_j \rangle \parallel \{f'_j \leftrightarrow l'_j\}))$$

When $\widetilde{l_1}, \widetilde{f_1}, \widetilde{l_2}, \widetilde{f_2}$ are clear from the context, we shall abbreviate $[\![N, \widetilde{l_1}, \widetilde{f_1}, \widetilde{l_2}, \widetilde{f_2}]\!]$ as $[\![N]\!]$. Intuitively, nodes in $\widetilde{l_1}$ are those whose scope must be captured by a half-restriction (see claim 1. of this Lemma), whereas nodes in $\widetilde{l_2}$ are those whose scope must be fully opened (see claim 2. of this Lemma). We now prove that the relation

$$\begin{aligned}\Re \triangleq \{(\,(\widetilde{l_1})N\,,\,(\widetilde{l_1})M\,)\ :\ & [\![N, \widetilde{l_1}, \widetilde{f_1}, \widetilde{l_2}, \widetilde{f_2}]\!] \cong [\![M, \widetilde{l_1}, \widetilde{f_1}, \widetilde{l_2}, \widetilde{f_2}]\!] \wedge \\ & (\widetilde{f_1}, \widetilde{f_2}) \cap fn(N, M, l) = \emptyset \ \wedge\ \widetilde{l_2} \subseteq fa(N) \cap fa(M)\}\end{aligned}$$

is a bisimulation up-to $\equiv$. Let $(\widetilde{l_1})N \xrightarrow{\alpha} \mathtt{N}$. Notice that, since $(\widetilde{f_1}, \widetilde{f_2})$ are reserved, they will remain fresh upon any transition; moreover, since free addresses can only increase upon transitions (see Proposition 6.2), nodes in $\widetilde{l_2}$ will remain present in any reduct of N and M. We reason by case analysis on α:[3]

1. $\alpha = \tau$**:** then, $\mathtt{N} \equiv (\widetilde{l_1})N'$, for $N \xrightarrow{\tau} N'$. This implies that $[\![N]\!] \xrightarrow{\tau} [\![N']\!]$; because of reduction closure of $\cong$, $[\![M]\!] \Rightarrow \bar{M}$ and $[\![N']\!] \cong \bar{M}$. Since $(\widetilde{f_1}, \widetilde{f_2})$ are fresh, it must be that $M \Rightarrow M'$ (thus, $(\widetilde{l_1})M \Rightarrow (\widetilde{l_1})M'$) and $\bar{M} \equiv [\![M']\!]$; hence, by definition, $\mathtt{N}\ \Re\ (\widetilde{l_1})M'$ up-to $\equiv$.

2. $\alpha = l \frown l'$**:** by definition of the LTS, $l \notin \widetilde{l_1}$; however, l' can belong to $\widetilde{l_2}$ or not, while it can be either $l \notin (\widetilde{l_1}, \widetilde{l_2})$, $l \in \widetilde{l_1}$ or $l \in \widetilde{l_2}$; moreover, if both l and l' belong to $\widetilde{l_2}$, we also have to consider whether they are the same name or not. This yield seven sub-cases:

   (a) $(\widetilde{l_1}, \widetilde{l_2}) \cap \{l, l'\} = \emptyset$**:** as before, $\mathtt{N} \equiv (\widetilde{l_1})N'$ for $N \xrightarrow{\alpha} N'$. Let $l_f \in (\widetilde{f_1}, \widetilde{f_2})$ (if $\widetilde{f_1}, \widetilde{f_2} = \emptyset$, we take a fresh l_f) and consider the context

   $$C[\cdot] \triangleq [\,\cdot\,] \parallel l' :: \mathbf{acpt}(l_f) \parallel l_f :: \textsc{Go}\ l'\ \textsc{Do}\ \mathbf{disc}(l)\ \textsc{Then}\ (\mathbf{nil} \oplus \mathbf{out}()@l_f)$$

   By context and reduction closure, we obtain that $(\widetilde{l_1})M \xRightarrow{l \frown l'} (\widetilde{l_1})M''$ and $(\widetilde{l_1})N'\ \Re\ (\widetilde{l_1})M''$.

   (b) $l \in \widetilde{l_1}$ **and** $l' \notin \widetilde{l_2}$**:** then $l \neq l'$ and $\mathtt{N} \triangleq (\widetilde{l'_1})N'$, for $N \xrightarrow{l \frown l'} N'$ and $\widetilde{l'_1} \triangleq \widetilde{l_1} - \{l\}$; let $l = l_i$ and f'_{h+1} be a new, reserved and fresh name. Consider the context

   $$\begin{aligned}C[\cdot] \triangleq\ & [\,\cdot\,] \parallel l' :: \mathbf{acpt}(f'_{h+1}) \parallel \{f_i \leftrightarrow f'_{h+1}\} \parallel \\ & f'_{h+1} :: \mathbf{in}(!x)@f_i.\mathbf{disc}(f_i).\textsc{Go}\ l'\ \textsc{Do}\ \mathbf{eval}(\mathbf{acpt}(f'_{h+1}))@x.\mathbf{disc}(x)\ \textsc{Then} \\ & \quad \mathbf{conn}(x).(\mathbf{nil} \oplus \mathbf{out}(x)@f'_{h+1})\end{aligned}$$

   By reasoning like above, we consider the reductions $C[[\![N]\!]] \models\!\Rightarrow \mathcal{D}'[[\![N', \widetilde{l'_1}, \widetilde{f_*}, \widetilde{l'_2}, \widetilde{f'_*}]\!]] \parallel f_i :: \mathbf{nil}$, where $\mathcal{D}'[\cdot]$ is defined similarly to case 2(a) above (with f'_{h+1} in place of l_f), $\widetilde{f_*} \triangleq \widetilde{f_1} - \{f_i\}$, $\widetilde{l'_2} \triangleq \widetilde{l_2} \cup \{l_i\}$ and $\widetilde{f'_*} \triangleq \widetilde{f_2} \cup \{f'_{h+1}\}$.

[3] We follow here a way of reasoning similar to the one used in Theorem 4.10. Thus, we shall only give the discriminating context $C[\cdot]$ and some details on the key issues.

(c) $l \in \widetilde{l_2}$ **and** $l' \notin \widetilde{l_2}$**:** let $l = l'_j$ and consider the context

$$\begin{aligned} C[\cdot] \triangleq\ [\,\cdot\,] \parallel l' :: \mathbf{acpt}(f'_j) \parallel f'_j :: \ & \mathbf{in}(!x)@f'_j.\textsc{Go}\ l'\ \textsc{Do}\ \mathbf{disc}(x)\ \textsc{Then} \\ & \mathbf{out}(x)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j) \end{aligned}$$

(d) $l' \in \widetilde{l_2}$ **and** $l \notin \widetilde{l_1}, \widetilde{l_2}$**:** like the previous case, with l in place of l'.

(e) $l \in \widetilde{l_1}$ **and** $l' \in \widetilde{l_2}$**:** let $l = l_i$, $l' = l'_j$ and f'_{h+1} be a new, reserved and fresh name; then, consider the context, derived from that in 2(b)

$$\begin{aligned} C[\cdot] \triangleq\ & [\,\cdot\,] \parallel \{f'_j \leftrightarrow f_i\} \parallel \{f'_j \leftrightarrow f'_{h+1}\} \parallel \\ & f'_j :: \mathbf{in}(!x)@f_i.\mathbf{disc}(f_i).\mathbf{in}(!y)@f'_j.\mathbf{out}(y)@f'_j.\mathbf{eval}(\mathbf{acpt}(f'_j))@y. \\ & \quad \textsc{Go}\ y\ \textsc{Do}\ \mathbf{eval}(\mathbf{acpt}(f'_{h+1}))@x.\mathbf{disc}(x)\ \textsc{Then} \\ & \quad \mathbf{eval}(\mathbf{disc}(f'_j).\mathbf{conn}(x).(\mathbf{nil} \oplus \mathbf{out}(x)@f'_{h+1}))@f'_{h+1} \end{aligned}$$

(f) $l \in \widetilde{l_2}$ **and** $l' \in \widetilde{l_2}$**, with** $l \neq l'$**:** let $l = l'_{j_1}$ and $l = l'_{j_2}$, for $j_1 \neq j_2$; consider the context

$$\begin{aligned} C[\cdot] \triangleq\ & [\,\cdot\,] \parallel \{f'_{j_1} \leftrightarrow f'_{j_2}\} \parallel \\ & f'_{j_2} :: \mathbf{in}(!y)@f'_{j_2}.\mathbf{in}(!x)@f'_{j_1}.\mathbf{eval}(\mathbf{acpt}(f'_{j_2}))@y. \\ & \quad \textsc{Go}\ y\ \textsc{Do}\ \mathbf{disc}(x)\ \textsc{Then}\ \mathbf{out}(y)@f'_{j_2}. \\ & \quad \mathbf{eval}(\mathbf{disc}(f'_{j_2}).\mathbf{out}(x)@f'_{j_1}.(\mathbf{nil} \oplus \mathbf{out}()@f'_{j_1}))@f'_{j_1} \end{aligned}$$

(g) $l = l' \in \widetilde{l_2}$**:** then, by rule (SELF), $l \in fa(M)$ implies $M \equiv M \parallel \{l \leftrightarrow l\}$, and the thesis easily follows.

3. $\alpha = \langle l\rangle @ l' : l'$ **:** by definition of the LTS, $l \notin \widetilde{l_1}$. Like case 2., we have seven subcases:

(a) $(\widetilde{l_1}, \widetilde{l_2}) \cap \{l, l'\} = \emptyset$**:** like case 2(a), with action $\mathbf{in}(l)@l'$ in place of $\mathbf{disc}(l)$ in $C[\cdot]$.

(b) $l = l_i$ **and** $l' \notin \widetilde{l_2}$**:** like case 2(c), with f_i in place of f'_j and $\mathbf{in}(x)@l'$ in place of $\mathbf{disc}(x)$.

(c) $l \in \widetilde{l_2}$ **and** $l' \notin \widetilde{l_2}$**:** like case 2(c), with $\mathbf{in}(x)@l'$ in place of $\mathbf{disc}(x)$.

(d) $l' = l'_j$ **and** $l \notin \widetilde{l_1}, \widetilde{l_2}$**:** consider the context

$$\begin{aligned} C[\cdot] \triangleq\ [\,\cdot\,] \parallel f'_j :: \ & \mathbf{in}(!x)@f'_j.\mathbf{eval}(\mathbf{acpt}(f'_j))@x. \\ & \textsc{Go}\ x\ \textsc{Do}\ \mathbf{in}(l)@x\ \textsc{Then}\ \mathbf{out}(x)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j) \end{aligned}$$

(e) $l = l_i$ **and** $l' = l'_j$**:** like case 2(f), with f_i in place of f'_{j_1}, f'_j in place of f'_{j_2} and $\mathbf{in}(x)@y$ in place of $\mathbf{disc}(x)$.

(f) $l = l'_{j_1}$ **and** $l = l'_{j_2}$**, for** $j_1 \neq j_2$**:** like case 2(f), with $\mathbf{in}(x)@y$ in place of $\mathbf{disc}(x)$.

(g) $l = l' \in \widetilde{l_2}$**:** like case 3(d), with x in place of l.

4. $\alpha = (\nu l)\,\langle l\rangle @ l' : l'$ **:** we have two subcases:

(a) $l' \notin \widetilde{l_2}$**:** then, $\mathtt{N} \equiv (\widetilde{l_1})\mathtt{N}'$, for $N \xrightarrow{\alpha} \mathtt{N}'$ and $\mathtt{N}' \equiv (l)\mathtt{N}''$. Let f_{k+1} be a new, reserved and fresh name; consider the context

$$C[\cdot] \triangleq\ [\,\cdot\,] \parallel \{f_{k+1} \leftrightarrow l'\} \parallel f_{k+1} :: \mathbf{in}(!x)@l'.\mathbf{disc}(l').\mathbf{out}(x)@f_{k+1}$$

Similarly to the 3rd case in the proof of Theorem 4.10, closure under such a context implies that $(\widetilde{l_1})M \overset{\alpha}{\Longrightarrow} (l, \widetilde{l_1})M'$ and $(l, \widetilde{l_1})\mathtt{N}'' \approx (l, \widetilde{l_1})M'$; we can easily conclude.

(b) $l' = l'_j$**:** as before, let f_{k+1} be a new, reserved and fresh name, and consider the context

$$C[\cdot] \triangleq [\,\cdot\,] \parallel \{f_{k+1} \leftrightarrow f'_j\} \parallel f'_j :: \mathbf{in}(!y)@f'_j.\mathbf{out}(y)@f'_j.\mathbf{in}(!x)@y. \\ \mathbf{eval}(\mathbf{disc}(f'_j).(\mathbf{out}(x)@f_{k+1} \oplus \mathbf{nil}))@f_{k+1}$$

5. $\alpha = \exists ?l \curvearrowright l'$**:** by definition of the LTS, $\{l, l\} \cap \widetilde{l_1} = \emptyset$; we have five subcases:

(a) $\{l, l'\} \cap \widetilde{l_2} = \emptyset$**:** we consider the context $C[\cdot] \triangleq [\,\cdot\,] \parallel \{l \leftrightarrow l'\}$ and easily conclude.

(b) $l = l'_j$ **and** $l' \notin \widetilde{l_2}$**:** consider the context

$$C[\cdot] \triangleq [\,\cdot\,] \parallel \{l' \leftrightarrow f'_j\} \parallel \\ f'_j :: \mathbf{in}(!x)@f'_j.\mathbf{eval}(\mathbf{acpt}(x))@l'.\mathbf{disc}(l').\mathbf{eval}(\mathbf{acpt}(f'_j))@x. \\ \text{GO } x \text{ DO } \mathbf{conn}(l') \text{ THEN } \mathbf{out}(x)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j)$$

(c) $l' = l'_j$ **and** $l \notin \widetilde{l_2}$**:** like case 5(b), with l in place of l'.

(d) $l = l'_{j_1}$ **and** $l = l'_{j_2}$**, for** $j_1 \neq j_2$**:** like case 2(f), with $\mathbf{acpt}(f'_{j_2}).\mathbf{acpt}(x)$ in place of $\mathbf{acpt}(f'_{j_2})$ and $\mathbf{conn}(x)$ in place of $\mathbf{disc}(x)$.

(e) $l = l' \in \widetilde{l_2}$**:** like case 2(g).

6. $\alpha = \; l : !l'$ **:** by definition of the LTS, $l \notin \widetilde{l_1}$; thus, like case 2., we have seven subcases:

(a) $(\widetilde{l_1}, \widetilde{l_2}) \cap \{l, l'\} = \emptyset$**:** like case 2(a), with $\mathbf{conn}(l).\mathbf{disc}(l)$ in place of $\mathbf{disc}(l)$.

(b) , (c) , (d) , (e) , (f)**:** like cases 2(b)/(c)/(d)/(e)/(f), with $\mathbf{conn}(x).\mathbf{disc}(x)$ in place of $\mathbf{disc}(x)$.

(g) like case 3(d), with $\mathbf{conn}(x).\mathbf{disc}(x)$ in place of $\mathbf{in}(l)@x$.

7. $\alpha = \; l : ?l'$ **:** like case 6., with **acpt** in place of **conn**.

8. $\alpha = (\nu l) \; l : ?l'$ **:** by definition of the LTS, $l \notin \widetilde{l_1}$; thus, we have two subcases:

(a) $l' \notin \widetilde{l_2}$**:** let f'_{h+1} be a new, reserved and fresh name; consider the context

$$C[\cdot] \triangleq [\,\cdot\,] \parallel \{f'_{h+1} \leftrightarrow l'\} \parallel l' :: \mathbf{acpt}(!x).\mathbf{eval}(\mathbf{conn}(f'_{h+1}))@x.\mathbf{disc}(x). \\ \mathbf{eval}(\mathbf{disc}(l').\mathbf{acpt}(x).(\mathbf{nil} \oplus \mathbf{out}(x)@f'_{h+1}))@f'_{h+1}$$

Notice that, by reasoning as in case 4(a), we can state that M performs a $\mathbf{conn}(l')$ from a restricted node (whose address can be alpha-converted to l).

(b) $l' = l'_j$**:** as before, let f'_{h+1} be a new, reserved and fresh name, and consider the context

$$C[\cdot] \triangleq [\cdot] \parallel \{f'_{h+1} \leftrightarrow f'_j\} \parallel f'_j :: \mathbf{in}(!y)@f'_j.\mathbf{out}(y)@f'_j.\mathbf{eval}(\mathbf{acpt}(f'_j))@y. \\ \text{GO } y \text{ DO } \mathbf{acpt}(!x).\mathbf{eval}(\mathbf{conn}(f'_{h+1}))@x.\mathbf{disc}(x) \text{ THEN} \\ \mathbf{eval}(\mathbf{disc}(f'_j).\mathbf{acpt}(x).(\mathbf{nil} \oplus \mathbf{out}(x)@f'_{h+1}))@f'_{h+1}$$

9. $\alpha = \exists ? \; \langle l \rangle @ l'' \; : \; l'$ **:** by definition of the LTS, $\{l, l''\} \cap \widetilde{l_1} = \emptyset$, while it can be $l \in \widetilde{l_1}$. By also distinguishing whenever $l = l'$, $l = l''$ and $l' = l''$, we have nineteen sub-cases:

(a) $\{l, l', l''\} \cap (\widetilde{l_1}, \widetilde{l_2}) = \emptyset$**:** consider the context $C[\cdot] \triangleq [\,\cdot\,] \parallel \{l' \leftrightarrow l''\} \parallel l'' :: \langle l \rangle$.

(b) $l = l_i$ **and** $\{l', l''\} \cap \widetilde{l_2} = \emptyset$**:** consider the context

$$\begin{aligned} C[\cdot] \triangleq\ & [\,\cdot\,] \parallel \{l'' \leftrightarrow f_i\} \parallel \{l'' \leftrightarrow l'\} \parallel \\ & f_i :: \mathbf{in}(!x)@f_i.\mathbf{out}(x)@f_i.\mathbf{out}(x)@l''.\mathbf{disc}(l'').(\mathbf{nil} \oplus \mathbf{out}()@f_i) \end{aligned}$$

(c) $l = l'_j$ **and** $\{l', l''\} \cap \widetilde{l_2} = \emptyset$**:** like case 9(b), with f'_j in place of f_i.

(d) $l' = l'_j$ **and** $\{l, l''\} \cap (\widetilde{l_1}, \widetilde{l_2}) = \emptyset$**:** consider the context

$$\begin{aligned} C[\cdot] \triangleq\ & [\,\cdot\,] \parallel \{l'' \leftrightarrow f'_j\} \parallel \\ & f'_j :: \mathbf{in}(!x)@f'_j.\mathbf{eval}(\mathbf{acpt}(x))@l''.\mathbf{out}(l)@l''.\mathbf{disc}(l'').\mathbf{eval}(\mathbf{acpt}(f'_j))@x. \\ & \quad \textsc{Go}\ x\ \textsc{Do}\ \mathbf{conn}(l'')\ \textsc{Then}\ \mathbf{out}(x)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j) \end{aligned}$$

(e) $l'' = l'_j$ **and** $\{l, l'\} \cap (\widetilde{l_1}, \widetilde{l_2}) = \emptyset$**:** consider the context

$$\begin{aligned} C[\cdot] \triangleq\ [\cdot] \parallel l' :: \mathbf{acpt}(f'_j) \parallel f'_j :: \ & \mathbf{in}(!x)@f'_j.\mathbf{eval}(\mathbf{acpt}(l'))@x.\mathbf{out}(l)@x. \\ & \textsc{Go}\ l'\ \textsc{Do}\ \mathbf{conn}(x)\ \textsc{Then} \\ & \mathbf{out}(x)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j) \end{aligned}$$

(f) $l = l_i$**,** $l' = l'_j$ **and** $l'' \notin \widetilde{l_2}$**:** consider the context, derived from that in case 9(d)

$$\begin{aligned} C[\cdot] \triangleq\ & [\,\cdot\,] \parallel \{l'' \leftrightarrow f'_j\} \parallel \{f'_j \leftrightarrow f_i\} \parallel \\ & f'_j :: \mathbf{in}(!y)@f_i.\mathbf{out}(y)@f_i.\mathbf{disc}(f_i). \\ & \quad \mathbf{in}(!x)@f'_j.\mathbf{eval}(\mathbf{acpt}(x))@l''.\mathbf{out}(y)@l''.\mathbf{disc}(l'').\mathbf{eval}(\mathbf{acpt}(f'_j))@x. \\ & \quad \textsc{Go}\ x\ \textsc{Do}\ \mathbf{conn}(l'')\ \textsc{Then}\ \mathbf{out}(x)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j) \end{aligned}$$

(g) $l = l'_{j_1}$, $l' = l'_{j_2}$, $j_1 \neq j_2$ **and** $l'' \notin \widetilde{l_2}$**:** like case 9(f), with f'_{j_1} in place of f_i and f'_{j_2} in place of f'_j.

(h) $l = l' = l'_j$ **and** $l'' \notin \widetilde{l_2}$**:** like case 9(d), with x in place of l.

(i) $l = l_i$**,** $l'' = l'_j$ **and** $l' \notin \widetilde{l_2}$**:** like case 9(f), with l' in place of l''.

(j) $l = l'_{j_1}$, $l'' = l'_{j_2}$, $j_1 \neq j_2$ **and** $l' \notin \widetilde{l_2}$**:** like case 9(g), with l' in place of l''.

(k) $l = l'' = l'_j$ **and** $l' \notin \widetilde{l_2}$**:** like case 9(e), with x in place of l.

(l) $l' = l'_{j_1}$, $l'' = l'_{j_2}$, $j_1 \neq j_2$ **and** $l \notin \widetilde{l_1}, \widetilde{l_2}$**:** consider the context

$$\begin{aligned} C[\cdot] \triangleq\ & [\,\cdot\,] \parallel \{f'_{j_1} \leftrightarrow f'_{j_2}\} \parallel f'_{j_1} :: \mathbf{acpt}(f'_{j_2}) \parallel \\ & f'_{j_2} :: \mathbf{in}(!x)@f'_{j_1}.\mathbf{out}(x)@f'_{j_1}.\mathbf{disc}(f'_{j_1}).\mathbf{in}(!y)@f'_{j_2}. \\ & \quad \textsc{Go}\ f'_{j_1}\ \textsc{Do}\ \mathbf{eval}(\mathbf{acpt}(y))@x\ \textsc{Then}\ \mathbf{eval}(\mathbf{acpt}(f'_{j_2}))@y. \\ & \quad \textsc{Go}\ y\ \textsc{Do}\ \mathbf{out}(l)@y.\mathbf{conn}(x)\ \textsc{Then}\ \mathbf{out}(y)@f'_{j_2}.(\mathbf{nil} \oplus \mathbf{out}()@f'_{j_2}) \end{aligned}$$

(m) $l' = l'' = l'_j$ **and** $l \notin \widetilde{l_1}, \widetilde{l_2}$**:** consider the context

$$C[\cdot] \triangleq [\,\cdot\,] \parallel f'_j :: \mathbf{in}(!y)@f'_j.\mathbf{out}(l)@y.\mathbf{out}(y)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j)$$

(n) $l = l_i$, $l' = l'_{j_1}$ **and** $l'' = l'_{j_2}$**, with** $j_1 \neq j_2$**:** consider the following context, derived from case 9(l):

$$\begin{aligned} C[\cdot] \triangleq\ & [\,\cdot\,] \parallel \{f_i \leftrightarrow f'_{j_2}\} \parallel \{f'_{j_1} \leftrightarrow f'_{j_2}\} \parallel f'_{j_1} :: \mathbf{acpt}(f'_{j_2}) \parallel \\ & f'_{j_2} :: \mathbf{in}(!z)@f_i.\mathbf{out}(z)@f_i.\mathbf{disc}(f_i). \\ & \quad \mathbf{in}(!x)@f'_{j_1}.\mathbf{out}(x)@f'_{j_1}.\mathbf{disc}(f'_{j_1}).\mathbf{in}(!y)@f'_{j_2}. \\ & \quad \textsc{Go}\ f'_{j_1}\ \textsc{Do}\ \mathbf{eval}(\mathbf{acpt}(y))@x\ \textsc{Then}\ \mathbf{eval}(\mathbf{acpt}(f'_{j_2}))@y. \\ & \quad \textsc{Go}\ y\ \textsc{Do}\ \mathbf{out}(z)@y.\mathbf{conn}(x)\ \textsc{Then}\ \mathbf{out}(y)@f'_{j_2}.(\mathbf{nil} \oplus \mathbf{out}()@f'_{j_2}) \end{aligned}$$

(o) $l = l_i$, $l' = l'' = l'_j$**:** consider the following context, derived from case 9(m):

$$\begin{aligned} C[\cdot] \triangleq\ [\,\cdot\,] \parallel \{f'_j \leftrightarrow f_i\} \parallel f'_j ::\ & \mathbf{in}(!z)@f_i.\mathbf{out}(z)@f_i.\mathbf{disc}(f_i).\mathbf{in}(!y)@f'_j. \\ & \mathbf{out}(z)@y.\mathbf{out}(y)@f'_j.(\mathbf{nil} \oplus \mathbf{out}()@f'_j) \end{aligned}$$

(p) $l = l'_j$, $l' = l'_{j_1}$ **and** $l'' = l'_{j_2}$**, with** $|\{j, j_1, j_2\}| = 3$**:** like case 9(n), with l'_j in place of l_i.

(q) $l = l' = l'_{j_1}$, $l'' = l'_{j_2}$ **and** $j_1 \neq j_2$**:** like case 9(l), with x in place of l.

(r) $l = l'' = l'_{j_2}$, $l' = l'_{j_1}$ **and** $j_1 \neq j_2$**:** like case 9(l), with y in place of l.

(s) $l = l' = l'' = l'_j$**:** like case 9(m), with y in place of l. ∎

Theorem 6.8 (Completeness) *If $N \cong M$ then $N \approx M$.*

Proof: We shall prove that the relation

$$\mathfrak{R} \triangleq \{\, (N, M) \ :\ N \cong M \,\} \cup \approx$$

is a bisimulation. Let $N \xrightarrow{\alpha} \mathtt{N}$ and reason by case analysis on α. Since the proof proceeds as that of Theorem 4.10, we shall only give the contexts used to force M to properly reply to α.

1. $\alpha = \tau$**:** the thesis easily follows from reduction closure.

2. $\alpha = \langle l \rangle @ l_1 : l_1$ **:** then $\mathtt{N} \triangleq N'$, for $N \xrightarrow{\alpha} N'$. By using the context exhibited in the corresponding case of Theorem 4.10, we get that $M \overset{\alpha}{\Longrightarrow} M'$ and, by Lemma 6.7(1), $(l')N' \approx (l')M'$; by (HEXT) and (HGARB), this yields $N' \approx M'$, as required.

3. $\alpha = (\nu l)\ \langle l \rangle @ l_1 : l_1$ **:** then, $\mathtt{N} \equiv (l)N'$, for $N \equiv (\nu l)N''$ and $N'' \xrightarrow{\langle l \rangle @ l_1 : l_1} N'$. Hence, we can proceed as in the corresponding case of Theorem 4.10 and obtain that $M \overset{\alpha}{\Longrightarrow} (l)M'$ and $(l)N' \approx (l)M'$, as required.

4. $\alpha = l_1 \curvearrowright l_2$**:** then $\mathtt{N} \triangleq N'$, for $N \xrightarrow{\alpha} N'$. Let l_f be a reserved and fresh name; now, consider the context

$$\begin{aligned} C[\cdot] \triangleq [\cdot] \parallel l_1 :: \mathbf{acpt}(l_f) \parallel l_f ::\ & \textsc{Go}\ l_1\ \textsc{Do}\ \mathbf{disc}(l_2)\ \textsc{Then} \\ & \mathbf{new}(l').\mathbf{disc}(l').(\mathbf{out}(l')@l_f \oplus \mathbf{nil}) \end{aligned}$$

   Like in case 2., we obtain $M \overset{\alpha}{\Longrightarrow} M'$ and $N' \approx M'$, as required.

5. $\alpha = \exists ?\beta$**:** the thesis easily follows by exploiting the context $C[\cdot] \triangleq [\,\cdot\,] \parallel \textsc{Net}(\beta)$.

6. $\alpha = l_2 : !l_1$ **:** consider the context

$$C[\cdot] \triangleq [\cdot] \parallel \{l_f \leftrightarrow l_1\} \parallel l_f :: \mathbf{conn}(l_2).\mathbf{disc}(l_2).\mathbf{disc}(l_1).\mathbf{new}(l').\mathbf{disc}(l')(\mathbf{out}(l')@l_f \oplus \mathbf{nil})$$

for l_f fresh, and proceed like in case 2.

7. $\alpha = l_2 : ?l_1$ **:** like case 6., with **acpt** in place of **conn**.

8. $\alpha = (\nu l_1)\ l_1 : ?l_2$ **:** then $N \triangleq N'$, for $N \equiv (\nu l_1)N''$ and $N'' \xrightarrow{l_1 : ?l_2} N'$. Let l_f be a reserved and fresh name; now, consider the context

$$C[\cdot] \triangleq [\cdot] \parallel l_2 :: \mathbf{acpt}(l_f) \parallel l_f :: \text{GO } l_2 \text{ DO } \mathbf{acpt}(!x).\mathbf{eval}(\mathbf{acpt}(l_f))@x.\mathbf{disc}(x) \text{ THEN } \mathbf{conn}(x).(\mathbf{out}(x)@l_f \oplus \mathbf{nil})$$

Like before, we obtain that $M \overset{\alpha}{\Longrightarrow} M'$; moreover, like in case 3., the node performing the **conn** must be bound also in M. Thus, by definition of the LTS, both N' and M' contain a free node with address l_1, since they both have performed a transition labelled with $(\nu l_1)\ l_1 : ?l_2$. By Lemma 6.7(2), this implies that $N' \approx M'$, as required. ∎

6.3 On a Trace-based Characterisation of May Testing

The theory in Section 5 can be smoothly adapted to the present setting. First, notice that label $\exists ?l$ is now replaced by label $(\nu \tilde{l})\ l_1 : ?l_2$. This allows us to simplify the complementation function as follows:

$$\overline{\beta} \triangleq \exists ?\beta \qquad \overline{\exists ?\beta} \triangleq \beta \qquad \overline{(\nu \tilde{l})\ l_2 : ?l_1} \triangleq l_1 : !l_2 \qquad \overline{l_2 : !l_1} \triangleq l_1 : ?l_2$$

The resulting equivalence is still a sound proof technique for may-testing (we leave the easy task of adapting the proofs in Section 5.2 to this new scenario). Unfortunately, we have not been able to prove the corresponding completeness result. The problem lies in the definition of canonical observers: to enable observations, other than providing some nodes, it is sometimes necessary to make such nodes accepting connection requests originating from `test`. While this task can be accomplished for free addresses, we have not found a smart and simple way to force restricted addresses to accept connection requests.

7 An Example: Dynamic Connections in a Cellular Net

In this section we model a scenario for communications between mobile devices and use the introduced proof techniques to analyze it and verify some relevant properties. Since we want to asses usability for practical purposes of the semantic theories for TKLAIM, we use the language of Section 2 for which we developed sound and complete proof techniques for both barbed congruence and may testing.

The scenario we consider is inspired by the *handover protocol*, proposed by the European Telecommunication Standards Institute (ETSI) for the GSM Public Land Mobile Network (PLMN). A formal specification and verification of the protocol by using the π-calculus can be found in [32]. The PLMN is a cellular system which consists of Mobile Stations (MSs), Base Stations (BSs) and Mobile Switching Centres (MSCs). MSs are mobile devices that provide services to end users. BSs manage the interface between the MSs and a stationary net; they control the communications within a geographical area (a cell). Any MSC handles a set of BSs; it communicates with them and with other MSCs using a stationary net. A *handover* occurs whenever the BS responsible for a MS should be changed during the computation (e.g., because the MS exit the area associated to the BS and entered in the area associated to a different BS).

We now model the handover of a PLMN in tKLAIM; for the sake of simplicity, we focus here on the aspects more closely related to connection handling; for more details, see [16]. Both MSs, BSs and MSCs are modelled as nodes. We shall exploit polyadic communications: thus, *tuples* of names will be used as basic data. Data will be retrieved by using *pattern matching*. A *pattern* is a sequence of names u and bound names $!x$. A pattern matches against a tuple if both have the same number of fi elds and corresponding fi elds match (i.e. two names match if they are identical, whereas a bound name matches any name). The pattern matching function in case of successful matching returns a substitution that associates the bound names of the pattern with the corresponding names of the tuple in the continuation process. All the theory we have developed in this paper for the monadic version of tKLAIM can be adapted to its polyadic version; the price to be paid is a heavier notation in the proofs (see, e.g., [22]).

We consider a simple PLMN, with one MSC (whose address is *msc*), n BSs (whose addresses are $bs_1, \ldots, bs_n$, resp.) and just one MS (whose address is l). We assume a private data repository of *msc*, located at the reserved node *table* and used to store two kinds of information: the address of the BSs (this is a permanent information) and the current MS-to-BS associations (that can change upon handover). The handover for l is handled by the MSC via the following process:

$$\begin{aligned} HNDVR \triangleq\ & \mathbf{in}(l,!x)@table.\mathbf{in}(!y)@table.\mathbf{out}(y)@table. \\ & \mathbf{eval}_x(\ \mathbf{disc}(l).\mathbf{eval}_{msc,y}(\ \mathbf{conn}(l).\mathbf{eval}_{msc}(\mathbf{out}(l,y)@table)\ )\ ) \end{aligned}$$

where $\mathbf{eval}_u(P)$ is a more readable way of writing process $\mathbf{eval}(P)@u$ and, similarly, $\mathbf{eval}_{u,v}(P)$ stands for $\mathbf{eval}(\mathbf{eval}(P)@v)@u$. Process *HNDVR* fi rst selects a MS-to-BS association to be changed (the reason why this is needed is not modelled here); then, it chooses a new BS, properly changes the connections between the MS and the BSs, and updates the repository *table*. By assuming that l is handled by the BS bs_i, the resulting system is

$$\begin{aligned} SYS_i \triangleq\ & (\nu\, table, bs_1, \ldots, bs_n)(msc :: *HNDVR \ \parallel\ \{msc \leftrightarrow table\} \ \parallel \\ & \prod_{j=1}^{n} (table :: \langle bs_j \rangle \ \parallel\ \{msc \leftrightarrow bs_j\}) \ \parallel\ table :: \langle l, bs_i \rangle \ \parallel\ \{bs_i \leftrightarrow l\}\ ) \end{aligned}$$

The main property we want to ensure in this scenario is that the MS l remains connected to the PLMN upon handovers. To formalise this requirement, we consider the following process:

$$CONN \triangleq \mathbf{in}(l,!x)@table.\mathbf{eval}_x(\ \mathbf{out}(\text{``}conn\text{''}, l, msc)@l.\mathbf{eval}_{\ msc}(\mathbf{out}(l,x)@table)\ )$$

Intuitively, this process aims at delivering to l a message stating that l is connected to the net governed by *msc*. Now, consider the following minor variation of SYS_i:

$$\begin{aligned} SYS_i' \triangleq\ & (\nu\, table, bs_1, \ldots, bs_n)(msc :: *HNDVR \mid *\, CONN \ \parallel\ \{msc \leftrightarrow table\} \ \parallel \\ & \prod_{j=1}^{n} (table :: \langle bs_j \rangle \ \parallel\ \{msc \leftrightarrow bs_j\}) \ \parallel\ table :: \langle l, bs_i \rangle \ \parallel\ \{bs_i \leftrightarrow l\}\ ) \end{aligned}$$

Soundness of the protocol can be established by proving that SYS_i' is behaviourally equivalent to *SPEC*, where

$$SPEC \triangleq msc :: \mathbf{nil} \ \parallel\ l :: *\mathbf{out}(\text{``}conn\text{''}, l, msc)@l$$

Intuitively, such an equivalence holds if (and only if) l is permanently connected to the net governed by *msc*; indeed, in *SPEC* we can produce at l as many data of the form $\langle \text{``}conn\text{''}, l, msc \rangle$ as wanted, whereas in SYS_i' this can be done only if there is always a connection between l and some BS.

Proof of soundness We shall give both a bisimulation-based and a trace-based proof of the soundness condition just described. To this aim, we define the following nets:

$$\begin{array}{rcl} PLMN & \triangleq & msc :: *HNDVR \mid *\,CONN \;\;\|\;\; \{msc \leftrightarrow table\} \\ & & \|\; \prod_{j=1}^{n} \;(table :: \langle bs_j\rangle \;\|\; \{msc \leftrightarrow bs_j\}) \\ PLMN_{-i} & \triangleq & msc :: *HNDVR \mid *\,CONN \;\;\|\;\; \{msc \leftrightarrow table\} \\ & & \|\; \prod_{j\neq i}^{1..n} \; table :: \langle bs_j\rangle \;\|\; \prod_{j=1}^{n} \;\{msc \leftrightarrow bs_j\} \end{array}$$

Intuitively, *PLMN* is the ‘static’ part of the net, i.e. the part (almost) always present in it; $PLMN_{-i}$ is a transient state of *PLMN* where the datum $\langle bs_i\rangle$ has been temporarily removed from *table*. If we let $\widetilde{l} \triangleq table, bs_1, \ldots, bs_n$, then we get that

$$SYS'_i \;=\; (\nu\widetilde{l})(PLMN \;\;\|\;\; table :: \langle l, bs_i\rangle \;\;\|\;\; \{bs_i \leftrightarrow l\}\,)$$

Now, let

$$l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \;\triangleq\; \begin{cases} l \;::\; \overbrace{\langle\text{“}conn\text{”}, l, msc\rangle \mid \cdots \mid \langle\text{“}conn\text{”}, l, msc\rangle}^{k} & \text{if } k > 0 \\ l \;::\; \mathbf{nil} & \text{if } k = 0 \end{cases}$$

Moreover, define also the following generalisations of SYS'_i and *SPEC*:

$$\begin{array}{rcl} SYS'_{i,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; table :: \langle l, bs_i\rangle \;\|\; \{bs_i \leftrightarrow l\}) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ SPEC_k & \triangleq & SPEC \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \end{array}$$

The following nets describe the evolutions of $SYS'_{i,k}$ arising from the execution of one copy of process *CONN*:

$$\begin{array}{rcl} N^0_{i,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; msc :: \mathbf{eval}_{bs_i}(\,\mathbf{out}(\text{“}conn\text{”}, l, msc)@l.\mathbf{eval}_{\,msc}(\mathbf{out}(l, bs_i)@table)\,) \\ & & \|\; \{bs_i \leftrightarrow l\}) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ N^1_{i,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; bs_i :: \mathbf{out}(\text{“}conn\text{”}, l, msc)@l.\mathbf{eval}_{\,msc}(\mathbf{out}(l, bs_i)@table) \;\|\; \{bs_i \leftrightarrow l\}) \\ & & \|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ N^2_{i,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; bs_i :: \mathbf{eval}_{msc}(\mathbf{out}(l, bs_i)@table) \;\|\; \{bs_i \leftrightarrow l\}) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k+1} \\ N^3_{i,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; msc :: \mathbf{out}(l, bs_i)@table \;\|\; \{bs_i \leftrightarrow l\}) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k+1} \end{array}$$

Similarly, the following nets describe the evolutions of $SYS'_{i,k}$ arising from the execution of one copy of process *HNDVR*:

$$\begin{array}{rcl} M^0_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; msc :: \mathbf{in}(!y)@table.\mathbf{out}(y)@table.\mathbf{eval}_{bs_i}(\cdots) \;\|\; \{bs_i \leftrightarrow l\}) \\ & & \|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^1_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN_{-j} \;\|\; msc :: \mathbf{out}(bs_j)@table.\mathbf{eval}_{bs_i}(\cdots) \;\|\; \{bs_i \leftrightarrow l\}) \\ & & \|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^2_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; msc :: \mathbf{eval}_{bs_i}(\mathbf{disc}(l).\mathbf{eval}_{msc,bs_j}(\cdots)) \;\|\; \{bs_i \leftrightarrow l\}) \\ & & \|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^3_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; bs_i :: \mathbf{disc}(l).\mathbf{eval}_{msc,bs_j}(\cdots) \;\|\; \{bs_i \leftrightarrow l\}) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^4_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; bs_i :: \mathbf{eval}_{msc,bs_j}(\mathbf{conn}(l).\mathbf{eval}_{msc}(\cdots))) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^5_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; msc :: \mathbf{eval}_{bs_j}(\mathbf{conn}(l).\mathbf{eval}_{msc}(\cdots))) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^6_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; bs_j :: \mathbf{conn}(l).\mathbf{eval}_{msc}(\cdots)) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^7_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; bs_j :: \mathbf{eval}_{msc}(\,\mathbf{out}(l, bs_j)@table\,) \;\|\; \{bs_j \leftrightarrow l\}) \\ & & \|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \\ M^8_{i,j,k} & \triangleq & (\nu\widetilde{l})(PLMN \;\|\; msc :: \mathbf{out}(l, bs_j)@table \;\|\; \{bs_j \leftrightarrow l\}) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^{\,k} \end{array}$$

We then consider the possible evolutions of nets $SYS'_{i,k}$, $SPEC_k$, $N^h_{i,k}$ and $M^h_{i,j,k}$.

$$
\begin{array}{rcll}
SYS'_{i,k} & \xrightarrow{msc \curvearrowright msc} & SYS'_{i,k} & \quad (1)\\
 & \xrightarrow{l \curvearrowright l} & SYS'_{i,k} & \quad (2)\\
 & \xrightarrow{\langle \text{“conn”}, l, msc\rangle\, @\, l:\, l} & SYS'_{i,k-1} \quad \text{for } k > 0 & \quad (3)\\
 & \xrightarrow{\tau} & N^0_{i,k} & \quad (4)\\
 & \xrightarrow{\tau} & M^0_{i,j,k} & \quad (5)\\
SPEC_k & \xrightarrow{msc \curvearrowright msc} & SPEC_k & \quad (6)\\
 & \xrightarrow{l \curvearrowright l} & SPEC_k & \quad (7)\\
 & \xrightarrow{\langle \text{“conn”}, l, msc\rangle\, @\, l:\, l} & SPEC_{k-1} \quad \text{for } k > 0 & \quad (8)\\
 & \xrightarrow{\exists ? l \curvearrowright l} & SPEC_{k+1} & \quad (9)\\
 & \xrightarrow{\tau} & SPEC_{k+1} & \quad (10)\\
N^h_{i,k} & \xrightarrow{msc \curvearrowright msc} & N^h_{i,k} & \quad (11)\\
 & \xrightarrow{l \curvearrowright l} & N^h_{i,k} & \quad (12)\\
 & \xrightarrow{\langle \text{“conn”}, l, msc\rangle\, @\, l:\, l} & N^h_{i,k-1} \quad \text{for } k > 0 & \quad (13)\\
 & \xrightarrow{\tau} & N^{h+1}_{i,k} \quad \text{for } h = 0, 2 & \quad (14)\\
 & \xrightarrow{\tau} & N^2_{i,k+1} \quad \text{for } h = 1 & \quad (15)\\
 & \xrightarrow{\tau} & SYS'_{i,k} \quad \text{for } h = 3 & \quad (16)\\
M^h_{i,j,k} & \xrightarrow{msc \curvearrowright msc} & M^h_{i,j,k} & \quad (17)\\
 & \xrightarrow{l \curvearrowright l} & M^h_{i,j,k} & \quad (18)\\
 & \xrightarrow{\langle \text{“conn”}, l, msc\rangle\, @\, l:\, l} & M^h_{i,j,k-1} \quad \text{for } k > 0 & \quad (19)\\
 & \xrightarrow{\tau} & M^{h+1}_{i,j,k} \quad \text{for } h < 8 & \quad (20)\\
 & \xrightarrow{\tau} & SYS'_{j,k} \quad \text{for } h = 8 & \quad (21)
\end{array}
$$

A bisimulation-based proof We must exhibit a bisimulation containing the pair $(SYS'_i, SPEC)$. Our candidate relation is

$$
\mathfrak{R} \triangleq \bigcup_{\substack{k \geq 0\\ i = 1..n}} \{(SYS'_{i,k}\,,\, SPEC_k)\} \;\cup\; \bigcup_{\substack{k \geq 0\\ i = 1..n\\ h = 0..3}} \{(N^h_{i,k}\,,\, SPEC_k)\} \;\cup\; \bigcup_{\substack{k \geq 0\\ i,j \in \{1..n\}\\ h = 0..8}} \{(M^h_{i,j,k}\,,\, SPEC_k)\}
$$

We now prove that it is indeed a bisimulation. Consider the pair $(SYS'_{i,k}\,,\, SPEC_k)$. The transitions (1), (2) and (3) are replied to by the transitions (6), (7) and (8) respectively, and vice versa; the transitions (4) and (5) are replied to by the empty sequence of τ actions; the transitions (9) and (10) are replied to by the sequence of τ actions (4), (14) and (15) leading $SYS'_{i,k}$ to $N^2_{i,k+1}$. Then, consider the pair $(N^h_{i,k}\,,\, SPEC_k)$. The transitions (11), (12) and (13) are replied to by the transitions (6),

(7) and (8) respectively, and vice versa; the transitions (14) and (16) are replied to by the empty sequence of τ actions; the transition (14) is replied to by the transition (10); the transitions (9) and (10) are replied to by the sequence of τ actions leading to $N^2_{i,k+1}$, if $h = 0, 1$, or by the sequence of τ actions $N^h_{i,k} \Rightarrow SYS'_{i,k} \Rightarrow N^2_{i,k+1}$, if $h = 2, 3$. Finally, consider the pair $(M^h_{i,j,k}\ ,\ SPEC_k)$. The transitions (17), (18) and (19) are replied to by the transitions (6), (7) and (8) respectively, and vice versa; the transitions (20) and (21) are replied to by the empty sequence of τ actions; the transitions (9) and (10) are replied to by the sequence of τ actions $M^h_{i,j,k} \Rightarrow SYS'_{j,k} \Rightarrow N^2_{j,k+1}$.

This proves that $\mathfrak{R}$ is a bisimulation; consequently, this suffices to prove that $SYS'_i \approx SPEC$, as $SYS'_i \equiv SYS'_{i,0}$ and $SPEC \equiv SPEC_0$.

A trace-based proof We must prove that any trace of SYS'_i can be replied to by a proper trace of $SPEC$, and vice versa. We start with the easier task, i.e. that $SPEC_k \overset{\sigma}{\Longrightarrow}$ implies that $SYS'_{i,k} \overset{\sigma'}{\Longrightarrow}$, for $\sigma' \preceq \sigma$. The proof is by induction on the length of σ; the base step is trivial. For the inductive step, let $\sigma \triangleq \phi \cdot \sigma_1$, i.e. $SPEC_k \overset{\phi}{\Longrightarrow} SPEC_{k'} \overset{\sigma_1}{\Longrightarrow}$. According to transitions (6)/.../(10), we have only four possibilities for the visible action ϕ:

$\phi = msc \curvearrowright msc$**:** then $k' \geq k$, as τ-actions can only expand the TS located at l in $SPEC_k$ (see transition (10)). By transitions (1), (4), (14), (15) and (16), $SYS'_{i,k} \overset{msc \curvearrowright msc}{\Longrightarrow} SYS'_{i,k'}$ and, by induction, there exists a $\sigma_2 \preceq \sigma_1$ such that $SYS'_{i,k'} \overset{\sigma_2}{\Longrightarrow}$. We can conclude, by letting σ' be $msc \curvearrowright msc \cdot \sigma_2$.

$\phi = l \curvearrowright l$**:** similar to the previous case.

$\phi = \langle \text{“}conn\text{''}, l, msc\rangle @ l : l$ **:** then $k' \geq k - 1$ and the proof proceeds like above.

$\exists ?l \curvearrowright l$**:** then $k' > k$. Thus, by transitions (4), (14), (15) and (16), we get $SYS'_{i,k} \Rightarrow SYS'_{i,k'}$. By induction, there exists a $\sigma_2 \preceq \sigma_1$ such that $SYS'_{i,k'} \overset{\sigma_2}{\Longrightarrow}$. We can conclude, by letting σ' be σ_2; indeed, by using law (L1), we have that $\sigma' \triangleq \sigma_2 \preceq \sigma_1 \preceq \phi \cdot \sigma_1 \triangleq \sigma$.

We now consider the converse. Actually, we prove a stronger result, i.e. that $SYS'_{i,k} \overset{\sigma}{\Longrightarrow}$ implies $SPEC_k \overset{\sigma}{\Longrightarrow}$. The proof is by induction on the length of σ; the base case is trivial. For the inductive case, let $\sigma \triangleq \phi \cdot \sigma'$, i.e. $SYS'_{i,k} \Rightarrow K \overset{\phi}{\rightarrow} K' \overset{\sigma'}{\Longrightarrow}$. If $K \triangleq SYS'_{i',k'}$, for some i' and $k' \geq k$ (again, τ-steps can only add data at l), the thesis follows by an easy induction. Otherwise, we have two possible sub-cases:

$K \triangleq M^h_{i',j,k'}$**:** then, $K' \triangleq M^h_{i',j,k''}$, where $k'' = k'$, if transitions (17) or (18) have been used, and $k'' = k' - 1$, if (19) has been used. In the first case, to be able to apply induction, we first need to let K' produce σ' through a net of the form $SYS'_{i',k'}$. Therefore, we consider the following alternative way[4] to produce σ': $K' \Rightarrow SYS'_{j,k'} \Rightarrow M^0_{j,i',k'} \Rightarrow SYS'_{i',k'} \Rightarrow M^h_{i',j,k'} \overset{\sigma'}{\Longrightarrow}$. Now, $SPEC_k \Rightarrow SPEC_{k'} \overset{\phi}{\rightarrow} SPEC_{k'}$ and, by induction, $SPEC_{k'} \overset{\sigma'}{\Longrightarrow}$; this suffices to conclude. In the second case, i.e. when transition (19) has been used, the proof can be carried out similarly: $K' \Rightarrow SYS'_{i',k'-1} \overset{\sigma'}{\Longrightarrow}$ and $SPEC_k \Rightarrow SPEC_{k'} \overset{\phi}{\rightarrow} SPEC_{k'-1} \overset{\sigma'}{\Longrightarrow}$.

[4]Notice that, since trace equivalence does not rely on co-induction (i.e., it is not reduction closed), the way in which $SYS'_{i,k}$ generates σ is not relevant.

$K \triangleq N^h_{i',k'}$: then, $K' \triangleq N^h_{i',k''}$, where $k'' = k'$, if transitions (11) or (12) have been used, and $k'' = k' - 1$, if (13) has been used. When $h = 0, 2, 3$, we can proceed exactly as in the previous case. When $h = 1$, we consider $K' \Rightarrow SYS'_{i',k''+1} \overset{\sigma'}{\Longrightarrow}$. Now, $SPEC_k \Rightarrow SPEC_{k'} \xrightarrow{\phi} SPEC_{k''} \Rightarrow SPEC_{k''+1}$ that, by induction, implies that $SPEC_{k''+1} \overset{\sigma'}{\Longrightarrow}$; again, this suffi ces to conclude.

Concluding remarks As this example should have pointed out, working with bisimilarity is definitely simpler than working with trace equivalence. To establish the former one, we only had to fi nd, for every action of one net, a proper reply of the other net. To establish the latter one, on the contrary, a more sophisticated reasoning was needed; indeed, trace equivalence is usually proved by inductive arguments, that makes it diffi cult to automatise. Moreover, even for a basic setting like CCS, bisimulation is tractable [26, 33] whereas trace equivalence is not [41]. We leave as a future work the task of adapting known algorithms and tools to (semi-)automatically work with our bisimulation.

8 Conclusions and Related Work

We have presented some semantic theories for TKLAIM, a process calculus equipped with primitives for process distribution and mobility, remote and asynchronous communication through distributed data repositories, dynamic activation and deactivation of inter-node connections. This combination of design choices has already proved to be valuable from both an implementative and applicative point of view. The semantic theories we introduced in this paper have been defi ned in a uniform fashion [7]: fi rst, we defi ned some basic observables for a global computing setting; second, we closed them under all possible contexts and/or reductions, thus obtaining two touchstone equivalences (namely *barbed congruence* and *may testing*); and third, we gave more tractable characterisations of these equivalences by means of *labelled bisimulation* and *trace equivalence*. The language proposed and its semantic theories have proved valuable to program and verify a non-trivial example, inspired by the *handover protocol*. Finally, we have also studied the impact on the semantic theories of adding a powerful primitive that enables a tight control on the activation of connections.

We believe that, although TKLAIM can be somewhat encoded in the π-calculus [14], the introduction of the former is justifi ed by at least two reasons. First, TKLAIM clearly enlightens the key features we want to model such as distribution and mobility of processes, and inter-node connections; an encoding of such features in the π-calculus would hide them within complex process structures. TKLAIM and π-calculus can be seen as formalisms standing at two different levels of abstraction: TKLAIM is *network aware* and allows the user to directly exploit knowledge of the topology of the net; the π-calculus (and more specifi cally its dialects more suitable for distributed implementations, like the Join calculus [18] and the $\pi_{1\ell}$-calculus [1]) is at network level and permits to directly refer network sockets (that can be represented by communication channels). Second, a convincing encoding should enjoy ‘reasonable’ properties, like those pointed out in [34]. We believe this is *not* the case. For example, in [14] we developed an intuitive encoding of a TKLAIM’s sub-calculus into the asynchronous π-calculus that does not preserve convergence. We are now working on proving that this is not incidental and is due to the check of existence of the target of a communication that is performed in TKLAIM and not in the π-calculus. We conjecture that a divergence free encoding does not exist.

We now conclude by touching upon some possible directions for future developments and upon most strictly related work.

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 ones. 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 [23].

Orthogonally, it would also be interesting to analyze effi ciency issues to better clarify, e.g., the advantages of mobile code and process distribution. A possible application is to fi nd out possible rearrangements of the processes over a given net that minimize the number of remote operations. In real scenarios, local operations are usually cheaper and faster than remote ones. A simple way to model this scenario is to assign costs to connections (see, e.g., [12]) and develop effi ciency preorders based on such information.

Related work Several calculi with process distribution and mobility have been proposed in the last decade. In the Introduction, we have already touched upon major differences between some calculi for global computers and TKLAIM from a linguistic point of view, namely the modelling of the network underlying global computers. Here, we want to mention work on equivalences for such languages.

To our knowledge, no alternative characterization of may testing in terms of a trace-based equivalence has ever been given for a distributed language with process mobility. On the contrary, bisimulation-based equivalences have been studied to some extent. Bisimulation-based equivalences for calculi relying on a flat net topology are developed in [1, 23]; such equivalences are mainly derived from bisimulation equivalences for the π-calculus and its variants. Bisimulation-based equivalences for calculi relying on a hierarchical net topology are developed in [28, 8, 10, 24]. Although these bisimulations are inspired by Sangiorgi's *context bisimulation* [37] and, thus, exploit universal quantifi cation over processes, they yield proof techniques that are usable in practice. Notice that the bisimulations introduced in the last two mentioned papers are sound but not complete proof techniques for the corresponding barbed congruences.

The work most closely related to ours is [20, 19]. There, a distributed version of the π-calculus, called Dπ_F, is presented where nodes are connected through links that can fail during the computation and a bisimulation-based proof technique is used to establish properties of systems. Indeed, they also tackle the problem of dealing with distributed systems whose behaviour is dependent on an underlying unreliable network, whose characteristics can vary over time and have to "manage" the knowledge about unreachable parts of the network. For this, they rely on partial views and 'unreachable nodes' while we resort to 'half-restricted' names. Their technical developments are however very different from ours: they need to resort to a type environment to take the knowledge of the net into account and nodes or links can only die (forever) while our **conn**/**acpt** primitives allow modeling of more dynamic networks.

References

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

[2] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

[3] L. Bettini, R. De Nicola, G. Ferrari, and R. Pugliese. Interactive Mobile Agents in X-KLAIM. In *Proc. of the 7th WETICE*, pages 110–115. IEEE Computer Society Press, 1998.

[4] L. Bettini, M. Loreti, and R. Pugliese. An Infrastructure Language for Open Nets. In *Proc. of SAC'02*, pages 373–377. ACM, 2002.

[5] M. Boreale and R. De Nicola. Testing equivalences for mobile processes. *Information and Computation*, 120:279–303, 1995.

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

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

[8] M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication and Mobility Control in Boxed Ambients. To appear in *Information and Computation*.

[9] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[10] G. Castagna and F. Zappa Nardelli. The Seal Calculus Revisited: contextual equivalence and bisimilarity. In *Proc. of FSTTCS*, volume 2556 of *LNCS*, pages 85–96. Springer, 2002.

[11] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In *Proc. of FSTTCS '98*, volume 1530 of *LNCS*, pages 90–101. Springer, 1998.

[12] R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A Process Calculus for QoS-Aware Applications. In *Proc. of COORDINATION'05*, number 3454 in LNCS, pages 33–48. Springer, 2005.

[13] 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.

[14] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of KLAIM-based calculi. To appear in *Theoretical Computer Science*.

[15] R. De Nicola, D. Gorla, and R. Pugliese. Basic observables for a calculus for global computing. In *Proc. of 32nd ICALP*, volume 3580 of *LNCS*, pages 1226–1238. Springer, 2005.

[16] R. De Nicola, D. Gorla, and R. Pugliese. Global computing in a dynamic network of tuple spaces. In *Proc. of COORDINATION'05*, number 3454 in LNCS, pages 157–172. Springer, 2005. Full version to appear in *Science of Computer Programming*.

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

[18] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *Proc. of CONCUR '96*, volume 1119 of *LNCS*, pages 406–421. Springer, 1996.

[19] A. Francalanza. A Study of Failure in a Distributed Pi-calculus. PhD Thesis, University of Sussex, 2005.

[20] A. Francalanza and M. Hennessy. A theory of system behaviour in the presence of node and link failures. Technical Report cs01:2005, Univ. of Sussex, 2005. An extended abstract appears in *Proc. of CONCUR'05*.

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

[22] D. Gorla. *Semantic Approaches to Global Computing Systems*. PhD thesis, Dip. Sistemi ed Informatica, Univ. di Firenze, 2004.

[23] 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 in *Theoretical Computer Science*.

[24] T. Hildebrandt, J. C. Godskesen, and M. Bundgaard. Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources. Technical Report ITU-TR-2004-52, IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark, Nov. 2004.

[25] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[26] P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes and Three Problems of Equivalence. *Information and Computation*, 86(1):43–68, 1990.

[27] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of POPL '00*, pages 352–364. ACM, 2000.

[28] M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. In *Proc. of ICALP'03*, volume 2719 of *LNCS*. Springer, 2003.

[29] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Information and Computation*, 100:1–77, Sept. 1992.

[30] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

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

[32] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4:497–543, 1992.

[33] R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[34] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[35] J. Parrow. An introduction to the pi-calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.

[36] 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).

[37] D. Sangiorgi. Bisimulation in higher-order process calculi. *Journal of Information and Computation*, 131:141–178, 1996.

[38] A. Schmitt and J.-B. Stefani. The M-calculus: a higher-order distributed process calculus. *SIGPLAN Not.*, 38(1):50–61, 2003.

[39] P. Sewell. From rewrite rules to bisimulation congruences. In *Proc. of CONCUR'98*, volume 1466 of LNCS, pages 269–284. Springer, 1998.

[40] P. Sewell, P. Wojciechowski, and B. Pierce. Location independence for mobile agents. In *Proc. of ICCL '98*, volume 1686 of *LNCS*. Springer, 1999.

[41] L. Stockmeyer and A. Meyer. Word Problems Requiring Exponential Time. In *Proc. of 5th Symp. on Theory of Computing (STOC)*, pages 1–9. ACM, 1973.