Information and Computation, 205(10):1491–1525. © *Elsevier, 2007.*

Basic Observables for a Calculus for Global Computing [★]

Rocco De Nicola [a] Daniele Gorla [b] Rosario Pugliese [a]

[a]*Dipartimento di Sistemi e Informatica, Università di Firenze*

[b]*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 has primitives 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. We then focus on barbed congruence and provide an alternative characterisation in terms of a labelled *bisimulation*. To test practical usability of the semantic theory, we model a system of communicating mobile devices and use the introduced proof techniques to verify one of its key properties.

Key words: Process calculi, distribution and mobility, explicit connections, basic observables, bisimulation, may testing

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

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

Email addresses: denicola@dsi.unifi.it (Rocco De Nicola), gorla@di.uniroma1.it (Daniele Gorla), pugliese@dsi.unifi.it (Rosario Pugliese).

by issues like communication, cooperation, mobility, resource usage, security, failure handling, etc.. We have thus witnessed to 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 describing the execution contexts where applications roam and run, for coordinating and monitoring resources usage, for modelling process communication and mobility, and for specifying and enforcing security policies.

In the last ten years, much research effort has been addressed to study the impact of different communication and mobility paradigms, but little attention has been devoted to modelling the actual network underlying global computers as such: it usually originates from the linguistic choices concerning the mobility paradigm. Some of the proposed foundational languages intend migration as the movement of bare processes; in this case, the network is seen as a fully connected graph of computing sites where new sites can be dynamically added (see, e.g., Dπ-calculus [22], KLAIM [13], $\pi_{1\ell}$-calculus [1] or NOMADIC PICT [35]). The remaining languages intend migration as the movement of entities with executable content (such as entire sites); in this case, the network is seen as a forest of trees that evolves by adding/pruning/moving subtrees (see e.g., Ambient [9] and its variants, DJoin [18], Homer [23], M-calculus [34], Seal [10]). However, global computers (e.g. the Internet) are generic graphs: their nodes are neither organised according to tree-like structures nor fully (directly) connected; moreover, connections can unpredictably break down rendering nodes (at least temporarily) unreachable.

To meet the demands arising from modelling the network topology of global computers and its evolution in time, in [16] we have introduced a new modelling language that takes its origin from two formalisms with opposite objectives, namely the programming language X-KLAIM [4,3] and the π-calculus [26]. The former one is a full fledged programming language based on KLAIM [13], whereas the latter one is the generally recognised 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 model the evolving interconnection structure underlying a network: TKLAIM provides three specific process primitives to activate, accept and deactivate inter-node connections. Connections become essential to perform remote operations; these are possible only if their source and target nodes are directly connected.

Here, we develop the semantic theory of TKLAIM and 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.

As possible basic observables, we have considered the following ones:

(*i*) a specific node is up and running (i.e., it provides a datum of any kind),
(*ii*) a specific information is available in (at least) a node,
(*iii*) a specific information is present at a specific node.

Other calculi for global computers make use of (barbed) congruences induced by similar observables; the barbs in Ambient are similar to (*i*), whereas those in Dπ-calculus are similar to (*iii*). Within our framework, it can be proved that, by closing observations under any TKLAIM context, the three basic observables all yield the same congruences. This is already an indication of the robustness of the resulting semantic theories. Moreover, the considered observables are sufficiently powerful to yield interesting semantic theories also when considering lower-level features, such as failures [16].

Of course, after defining equivalences as context closures, it is essential to determine alternative characterisations that permit a better appreciation of their discriminating power and to devise proof techniques that avoid universal quantification over contexts, thus simplifying equivalence checking. For more standard process languages, barbed congruence is characterised via a bisimulation-based equivalence, whereas may testing is characterised via a trace-based equivalence. In [15], we developed such characterisations for a simplified version of TKLAIM; however, even for that simpler language, it turned out that trace equivalence was very complex and gave (almost) no insight into the discriminating power of may testing. In this paper, we thus focus on the bisimulation-based characterisation of barbed congruence. More precisely, we study the barbed congruence induced by the basic observable (*i*) above and define its alternative characterisation in terms of a labelled *weak bisimilarity*.

To this aim, we rely on a *labelled transition system* (LTS) simpler than the one in [15] and whose distinctive feature is that labels indicate the resources a term offers or requires to the execution context for combined evolution. We then define weak bisimilarity on top of this LTS and present soundness and completeness results with respect to barbed congruence. The actual development of the alternative characterisation, although performed along the lines of similar results for CCS [27] and π-calculus [32,2], had to face problems induced by process distribution and mobility, by asynchrony and by the explicit modelling of connections.

To gently introduce the reader to the technicalities of our theory, we start by developing it for the simplified version of TKLAIM presented in [15], where success of a connection request does not depend on acceptance by the partner. Then, we move to a more sophisticated (and realistic) framework where requests for connection activations must be authorised by the target node. The main difference between the LTS of the simplified and of the full language is the way in which scope of restricted names is opened. In the simplified language, we work like in π-calculus: if the address of a restricted node is transmitted as a datum (the name is *extruded*), the scope of such name is opened; thus, from the point of view of a receiving process, the node at such address becomes indistinguishable from a free node (the process can freely connect to the restricted node and access it). For the full language, scope handling becomes more critical. Extrusion alone does not suffice to open the scope of a restricted name: if the node is 'unreachable' (i.e., it is not connected with any other node of the net and does not accept/require connections), no receiving process can have access to it. Thus, extruding a name does not completely remove the restriction. For this reason, we introduce the notion of *half-restrictions* and model opening of the scope in two steps: firstly, a restricted name becomes half-restricted; then, whenever the half-restricted node exhibits/accepts/requires a connection with another node of the net, the half-restriction is removed and the scope is fully opened.

The rest of the paper is organised as follows. In Section 2, we present TKLAIM's syntax and reduction-based semantics and, in Section 3, we define barbed congruence and may testing for it. In Section 4, we consider the simplified version of TKLAIM and develop the alternative characterisation of barbed congruence for it; in Section 5, we then adapt the theory to the full language. In Section 6, we briefly present some sound trace-based laws that can be used to establish may testing and discuss the difficulties of characterising the latter equivalence when considering the language with connection acceptance. In Section 7, we present an example illustrating the use of TKLAIM and of its semantic theories to state and prove properties of global computing applications. In particular, we prove the same equality by using both bisimulation and trace equivalence to compare the two approaches. Finally, in Section 8, we draw some conclusions and briefly discuss some related work.

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.

TKLAIM [16] adopts a LINDA-like [20] asynchronous communication mechanism: interaction between (sending and receiving) processes are mediated by network nodes acting as repositories for anonymous data tuples which are retrieved through pattern-matching. Here, to mitigate the technicalities, we consider a minor variant of the language in [16] with only monadic data and with process replication in place of process definitions. Nevertheless, all the semantic theories we shall develop could be smoothly extended to the original language.

The syntax of TKLAIM is reported in Table 1, where we assume 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

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 ::=	$\mathbf{in}(!x)@u \;\big|\; \mathbf{in}(u_2)@u_1 \;\big|\; \mathbf{out}(u_2)@u_1 \;\big|\; \mathbf{eval}(P)@u$
		$\big|\; \mathbf{new}(l) \;\big|\; \mathbf{conn}(u) \;\big|\; \mathbf{disc}(u) \;\big|\; \mathbf{acpt}(!x) \;\big|\; \mathbf{acpt}(u)$

Table 1
tKlaim Syntax

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.

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**), sending processes for (possibly remote) execution (action **eval**), creating new nodes (action **new**), and activating/deactivating/accepting connections (actions **conn**, **disc** and **acpt**). 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 retrieves any datum. Indeed, $\mathbf{in}(l)@l'$ is a form of *name matching operator* reminiscent of Linda's pattern-matching [20]. A similar difference holds for action **acpt**: $\mathbf{acpt}(l)$ only accepts connection requests coming from l, whereas $\mathbf{acpt}(!x)$ accepts connections from any node. In Table 1, we have highlighted the **acpt** construct to stress that, for the sake of presentation, it is not part of the simplified variant of the language examined in Section 4.

[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.

Names occurring in tKlaim processes and nets can be *bound*. More precisely, prefixes $\mathbf{in}(!x)@u.P$ and $\mathbf{acpt}(!x).P$ bind 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. 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 a term $\_$ if $u \notin n(\_)$. In the sequel, we shall work with terms whose bound names are all distinct and different from the free ones.

Notation 2.1 We write $A \triangleq W$ to mean that A is of the form W; this notation is used to assign a symbolic name A to the term W. We shall use notation $\widetilde{\cdot}$ to denote 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 trailing 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.

To conclude the presentation of tKlaim's syntax, let us discuss on its peculiar primitives, namely those for handling connections. Firstly, notice that connection activations must be authorised by a corresponding **acpt**. We have not introduced an enabling action for disconnections as these are usually unilateral (asynchronous) events. Secondly, for action **acpt**, like for **in**, we have two variants (assume that the invoking process is running at l'): $\mathbf{acpt}(l)$ means that l' is ready to activate a connection with l, whereas $\mathbf{acpt}(!x)$ means that l' is ready to activate a connection with any node, whose address will replace x in the continuation. $\mathbf{acpt}(!x)$ can be exploited by a server willing to accept connection requests from any, initially unknown, client. On the other hand, $\mathbf{acpt}(l)$ should be used if a process is ready to activate connections only with a specific partner. One could think of simulating $\mathbf{acpt}(l)$ by accepting connection requests from any process through $\mathbf{acpt}(!x)$ and then, after checking the partner identity, disconnecting the unwanted partners through **disc**. But this could expose a node to security risks because the sequence of actions is not guaranteed to be performed atomically. It is also worth noting that the form of client-server interaction enabled by $\mathbf{acpt}(!x)$ could not be flexibly implemented by resorting to a shared repository storing connection requests, because a connection between the node hosting the repository and that of a potential client should be already in place for the client to be able to put its request.

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 2. The first eight laws are borrowed from the π-calculus (see, e.g., [31]); the rest of the laws have the following meaning. Law (Abs) is the equivalent

(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)	$\{l_1 \leftrightarrow l_2\} \equiv \{l_2 \leftrightarrow l_1\}$
(CONN)	$\{l_1 \leftrightarrow l_2\} \equiv l_1 :: \mathbf{nil} \parallel \{l_1 \leftrightarrow l_2\}$		

Table 2
Structural Congruence

of law (PZERO) for '|'. Law (CLONE) transforms a parallel between co-located components into a parallel between nodes (together with laws (PCOM) and (PASS), it implies that '|' is a commutative and associative operator). Laws (SELF), (BIDIR) and (CONN) 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 are placed only between existing nodes.

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.

The *reduction relation* is given in Table 3. We briefly comment on some crucial points. In (R-OUT) and (R-EVAL), existence of a connection between the nodes source and target of the action is necessary to place the spawned component. Rules (R-IN) and (R-MATCH) additionally require 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. Rule (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. Rule (R-DISC) deals with deactivation of connections and checks existence of the connection to be deactivated in order to execute the action. Finally, rules (R-CONN), (R-CONN$_1$) and (R-CONN$_2$) deal with activation of connections. The first rule is used when the **acpt** construct is not required; in this case, only existence of the nodes that are being connected is checked. The remaining two rules are used to synchronise a **conn** with one of the two variants of **acpt**.

If $N \longmapsto N'$, we shall say that N can perform a reduction step and that N' is a *reduct*

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

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

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

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

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

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

(R-PAR) $$\frac{N_1 \longmapsto N_1'}{N_1 \parallel N_2 \longmapsto N_1' \parallel N_2}$$ (R-RES) $$\frac{N \longmapsto N'}{(\nu l)N \longmapsto (\nu l)N'}$$ (R-STRUCT) $$\frac{N \equiv M \longmapsto M' \equiv N'}{N \longmapsto N'}$$

TKLAIM *without the* **acpt**:

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

Full TKLAIM *(with the* **acpt***)*:

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

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

Table 3
TKLAIM Operational Semantics

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 introduce 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 net.

Definition 3.1 (Observables or barbs)

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

The first observable above corresponds to that labelled (i) in the Introduction and is to some extent inspired from that of 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.

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 $\mathcal{C}[\cdot]$ *is a* TKLAIM *net with a hole* $[\cdot]$ *to be filled with any net. Formally,*

$$\mathcal{C}[\cdot] \;::=\; [\cdot] \;\;\Big|\;\; N \,\|\, \mathcal{C}[\cdot] \;\;\Big|\;\; (\nu l)\, \mathcal{C}[\cdot]$$

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

Any reasonable equivalence should of course 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.*

From their definitions, it trivially follows that $\cong$ is contained in $\simeq$. The inclusion is strict because the latter equivalence abstracts from the branching structure of the equated nets, whereas the former one does not (because of reduction closure). Thus, it is easy to prove the following result.

Proposition 3.6 $\cong \;\subset\; \simeq$.

The above definition of (reduction) barbed congruence is the standard one [24]; may testing is, instead, usually defined in terms of *observers*, *computations* and *success of a computation* [17]. In Section 6 we prove that such an alternative characterisation can be given for $\simeq$.

The problem with the definitions of barbed congruence and may testing is that context closure makes it difficult to prove equivalences due to the universal quantification over contexts. In the following sections, we shall provide an alternative characterisation of $\cong$ as a *bisimulation-based* equivalence, both for the simplified and for the full version of TKLAIM; moreover, we shall also present a sound trace-based proof-technique for may testing.

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.

Proposition 3.7 (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 $\{l, l'\} \cap \widetilde{l} = \emptyset$

Then, $\cong_1 = \cong_2 = \cong$ 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 for TKLAIM without 'acpt'

In this section, we provide a more tractable characterisation of barbed congruence by means of a *labelled bisimulation* for TKLAIM without the **acpt** primitive. To this aim, we start by presenting an alternative semantics (still equivalent to the one induced by the reductions of Table 3) 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

The labelled transition system (LTS) makes apparent the possible contributions that a net offers/requires in a computation. The *labelled transition relation*, $\xrightarrow{\alpha}$, is defined as the least relation over nets induced by the inference rules in Table 4. Labels take the form

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

In the sequel, we shall write $(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_2$ to denote $\langle l\rangle @ l_1 : l_2$, if $\widetilde{l} = \emptyset$, and $(\nu l)\ \langle l\rangle @ l_1 : l_2$, otherwise (i.e., if $\widetilde{l} = \{l\}$). Moreover, we let $bn(\alpha)$ be $\{l\}$, if $\alpha = (\nu l)\ \langle l\rangle @ l_1 : l_2$, 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 **:** N may perform a reduction step to become N' (see Proposition 4.2).

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

$(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_2$ **:** a datum $\langle l\rangle$ located at l_1 is offered to processes 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 the offering net or not (see (LTS-OPEN)).

$\exists?l_1 \curvearrowright l_2$**:** there is a process located at l_1 that needs a connection with l_2 (see rules (LTS-DISC), (LTS-OUT) and (LTS-EVAL)). In case $l_1 = l_2$, the sole existence of node l_2 is required (see rule (LTS-CONN) and the structural rule (SELF)). In both cases, such requirement is fulfilled by a ‘complementary’ label $l_1 \curvearrowright l_2$ (see rule (LTS-COMPL)).

$\exists?\ \langle l\rangle @ l_2 : l_1$ **:** 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, such a requirement must be satisfied by label $\langle l\rangle @ l_2 : l_1$ (see (LTS-COMPL)).

To briefly sum up, labels of the form $l_1 \curvearrowright l_2$ and $(\nu\widetilde{l})\ \langle l\rangle @ l_1 : l_2$ provide information about the structure of a net and about the resources (connections and data) the net can ‘offer’ to the execution context for combined evolution. On the other hand, labels of the form $\exists?\beta$ indicate the resources a net ‘demands’ to the execution context for combined evolution. Thus, (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 an intention is concretised, l_1 will be left with process P, l_2 will receive the datum $\langle l\rangle$, and the connection $\{l_1 \leftrightarrow l_2\}$, provided by the execution context, will be

(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 4
A Labelled Transition System for TKLAIM without ‘**acpt**’

left for future use”. Indeed, since label $\exists ? l_1 \curvearrowright l_2$ requires existence of the connection $\{l_1 \leftrightarrow l_2\}$, every execution context satisfying this requirement allows the sending net to place the datum at the target node and to assume existence of the connection needed. 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**, **disc** and **conn**), instead of having a different label for each possible action. This permits simplifying all the 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., [31]), 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.

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 $N \xrightarrow{\alpha} N' \xrightarrow{\alpha'} M$. As usual, we let $\Rightarrow$ stand for $\xrightarrow{\tau}{}^*$ and $\xRightarrow{\alpha}$ to stand for $\Rightarrow\xrightarrow{\alpha}\Rightarrow$; finally, $\xRightarrow{\hat{\alpha}}$ denotes $\Rightarrow$, if $\alpha = \tau$, and $\xRightarrow{\alpha}$, otherwise.

We conclude the presentation of the LTS by presenting some of its properties. First, we show that the LTS is 'correct' w.r.t. the operational semantics of TKLAIM based on $\longmapsto$. Then, we connect transitions offering resources with the syntactical form of the net performing them. Finally, 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.

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

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

Proposition 4.3 *The following facts hold:*

(1) if $N \xrightarrow{l_1 \frown l_2} N'$, 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 $l \notin \{l_1, l_2\}$.

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

Proposition 4.4 $(\nu\widetilde{l})(N \parallel K) \xrightarrow{\alpha} \bar{N}$ *if and only if one of the following conditions holds, possibly exchanging K and N:*

(1) $(\nu\widetilde{l})N \xrightarrow{\alpha} (\nu\widetilde{l'})N'$ and $\bar{N} \equiv (\nu\widetilde{l'})(N' \parallel K)$.

(2) $N \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle\,@\,l_1 : l_1} N'$, $K \xrightarrow{l_1 \curvearrowright l_2} K'$ *and* $\bar{N} \equiv (\nu\widetilde{l''})(N' \parallel K')$, *where* $\alpha = (\nu l)\ \langle l\rangle\ \underline{@}\ l_1 \underline{:}\ l_2$ *and* $\widetilde{l''} = \widetilde{l} - \{l\}$, *if* $\widetilde{l'} = \emptyset$ *and* $l \in \widetilde{l}$, *whereas* $\alpha = (\nu\widetilde{l'})\ \langle l\rangle\ @\ l_1 : l_2$ *and* $\widetilde{l''} = \widetilde{l}$, *otherwise.*

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

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

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

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

Proof: The "if" part is trivial, by using the LTS of Table 4. 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 2. In this proof, (LTS-STRUCT) always refers 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). For the details, see Appendix 8. ∎

4.2 A Bisimulation-based Characterisation of Barbed Congruence

We can now introduce the alternative characterisation of $\cong$ in terms of a labelled *bisimilarity*. To this aim, we first define the 'minimal' net NET(β) enabling the evolution of a net performing a label of the form $\exists ?\beta$; formally,

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

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

(1) *if* $\alpha \in \{\tau, l_1 \curvearrowright l_2, (\nu\widetilde{l})\ \langle l\rangle\ @\ l_1 : l_1\}$, *then* $M \overset{\hat{\alpha}}{\Longrightarrow} M'$ *and* $N' \Re M'$, *for some* M';

(2) *if* $\alpha = \exists ?\beta$, *then* $M \parallel \mathrm{NET}(\beta) \Rightarrow M'$ *and* $N' \Re M'$, *for some* M'.

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

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

Labels of the form $\exists?\beta$ are 'requirements' to the execution context; thus, they are handled differently. For example, requiring existence of a connection (e.g., to send a component) expressed by $N \xrightarrow{\exists?l_1\frown l_2} N'$ can be simulated by a net M in a context where l_1 and l_2 are connected through the execution of 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.

The crucial step to prove that bisimilarity is a sound proof-technique for barbed congruence is Lemma 4.7. 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; this is an easy task, by using law (LTS-STRUCT). Then, 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 the relation

$$\mathfrak{R} \triangleq \{ \left((\nu\widetilde{l})(N \parallel K), (\nu\widetilde{l})(M \parallel K) \right) \ : \ N \approx M \ \text{ and } K \text{ restriction free} \}$$

is a bisimulation up-to $\equiv$ (recall that, by (LTS-STRUCT), any context $\mathcal{C}[\cdot]$ is structurally equivalent to $(\nu\widetilde{l})(\ [\cdot]\ \parallel K)$, for a proper $\widetilde{l}$ that makes K restriction free). We must show that every transition from $(\nu\widetilde{l})(N \parallel K)$ can be 'matched' (according to Definition 4.5) by a transition from $(\nu\widetilde{l})(M \parallel K)$ and that the resulting nets are still in $\mathfrak{R}$. According to Proposition 4.4, we have twelve possible cases for the first transition; this case analysis is omitted, as it can be inferred from the proof of (the similar, but more complicated) Lemma 5.7. ▮

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

$N \xRightarrow{(\widetilde{\nu l})\ \langle l' \rangle @ l : l}$, for some l' and $\widetilde{l}$. By hypothesis, we get that $M \xRightarrow{(\widetilde{\nu l})\ \langle l' \rangle @ l : l}$; thus, by Proposition 4.3(1), $M \Downarrow l$. To prove that $\approx$ is reduction closed, let $N \longmapsto N'$; by Proposition 4.2, 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 4.2, $M \models\!\Rightarrow M'$. ∎

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

Notation 4.9 We write

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

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

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

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

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

It is easy to see that $l' :: P \oplus Q$ behaves as either $l' :: P$ or $l' :: Q$ and these possibilities are mutually exclusive.

The following key Lemma states that we can throw away a fresh locality hosting a restricted datum from two barbed congruent nets and the resulting nets, deprived of the restriction, are bisimilar.

Lemma 4.10 *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

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

is a bisimulation up-to $\equiv$. We omit the details of the proof because it proceeds as the (more complicated) proof of Lemma 5.8. ∎

Theorem 4.11 (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.

$\alpha = \tau$. By Proposition 4.2, 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' and l_f fresh, and the reduction $\mathcal{C}[N] \models\!\Rightarrow \mathcal{D}[N'] \triangleq \bar{N}$, where

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

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

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

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 $\bar{M} \models\!\Rightarrow \bar{M}'$ and $\bar{N}' \cong \bar{M}'$; because of freshness of l_f, this implies that $\bar{M}' \equiv \mathcal{D}'[M'']$, 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, $(\nu l')(N' \parallel l_f : \langle l'\rangle) \cong \mathcal{D}'[N'] \triangleq \bar{N}' \cong \bar{M}' \equiv \mathcal{D}'[M''] \cong (\nu l')(M'' \parallel l_f : \langle l'\rangle)$. Since $\equiv \subseteq \cong$ and since $\cong$ is transitive, by Lemma 4.10, we get $N' \approx M''$; this suffices to conclude.

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

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

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 $\mathcal{C}[M]$, whereas l_f exhibits a restricted datum in the chosen evolution of $\mathcal{C}[N]$.[2] Such a restricted datum can be then 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

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

and proceed like before.

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

∎

[2] To see that there must be an evolution of $\mathcal{C}[M]$ 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 chosen evolution of $\mathcal{C}[N]$ will never enable the production of a datum at l'_f because we assumed that bound names are different from the free ones; thus, M cannot produce only free data at l_1, otherwise it would not be equivalent to N.

From Theorems 4.8 and 4.11 we get the wanted result.

Corollary 4.12 (Alternative Characterisation of Barbed Congruence) $\approx \, = \, \cong$.

5 Bisimulation Equivalence for tKlaim

In this section, we develop a bisimulation-based characterisation of barbed congruence for tKlaim, i.e. for the language where a handshake between the nodes involved is necessary to activate a connection. We first reformulate the LTS presented in Section 4.1 to deal with the finer scenario we are investigating now; then, we move to the alternative characterisation of barbed congruence, by smoothly adapting Definition 4.5.

5.1 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:

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

Intuitively, $(\nu\widetilde{l})\, l_1 \ : ?l_2$ results from enriching label $\exists ?l_2 \curvearrowright l_2$ (that in the LTS of Section 4.1 labels transitions due to action **conn**(l_2)) with the node address $\widetilde{l_1}$ (that can also be restricted) where the action takes place; of course, $bn(\ (\nu\widetilde{l})\, l_1 \ : ?l_2 \ ) \triangleq \widetilde{l}$. Label $l_1 \ : !l_2$ instead indicates execution at l_1 of an action **acpt** accepting a connection request from l_2. These new labels are generated by rules (LTS-Conn), (LTS-Acc$_1$) and (LTS-Acc$_2$) in Table 6; they are synchronised via rule (LTS-Est), which establishes a new connection as a consequence of a synchronisation between a connection request and an acceptance. Like for (LTS-Compl), no scope extension 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 6 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

$$(\nu l')\,(l :: \langle l' \rangle \,||\, l' :: C)$$

It would be too informative to state that $(\nu l')\,(l :: \langle l' \rangle \,||\, l' :: C) \xrightarrow{(\nu l') \ \langle l' \rangle @ l : l} l ::$ **nil** $||\, l' :: C$. Indeed, if $C \triangleq \langle\rangle$, no context can observe the datum at l', because

(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 $l \notin fa(\mathtt{N})$

Table 5
Additional Structural Congruence Laws

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

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

are barbed congruent. However, by fully opening the scope of l', the bisimilarity would be able to distinguish these two nets. Thus, the latter equivalence would have an observational power that no language context actually has; this would mean that barbed congruence does not imply bisimilarity, that would invalidate completeness.

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 that may possibly contain *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 since l is known to the environment (that has previously received the name via an action generating label $(\nu l)\ \langle l \rangle @ l_1 : l_2$); thus, $fn((l)\mathtt{N}) \triangleq \{l\} \cup fn(\mathtt{N})$ and, in $(l)\mathtt{N}$, l cannot be alpha-converted.

To cope with half-restrictions, the structural congruence $\equiv$ defined through the laws in Table 2 is extended with the laws in Table 5. In (HEXT), we write $l \in fa(N)$ if $N \equiv N \parallel l :: \mathbf{nil}$, 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 $\mathtt{N}$ because it is still (potentially) unreachable in $\mathtt{M}$.

Now, a transition with label $(\nu l)\ \langle l \rangle @ l_1 : l_2$ is performed when a restriction on l is turned to a half-restriction, see rule (LTS-HALFOPEN) in Table 6. 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

(LTS-ACC$_1$)

$$l_1 :: \mathbf{acpt}(l_2).P \xrightarrow{l_1 : !l_2} l_1 :: P$$

(LTS-ACC$_2$)

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

(LTS-CONN)

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

(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)

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

(LTS-HALFRES$_1$)

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

(LTS-HALFRES$_2$)

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

plus all rules from Table 4, but (LTS-CONN) and (LTS-OPEN), with $\mathtt{N}$ in place of N everywhere and with the extended $\equiv$.

Table 6
The Revised Labelled Transition System for full TKLAIM

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 because only some actions involving it are forbidden.

Summarising, the revised labelled transition relation (between extended nets) is presented in Table 6. Notably, its τ-steps still coincide with the reductions of the extended language. That is, the following analogous of Propositions 4.2 holds.

Proposition 5.1 *$N \longmapsto M$ if and only if $N \xrightarrow{\tau} M$.*

Also, an analogous of Proposition 4.3 holds.

Proposition 5.2 *The following facts hold:*

(1) *if* $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{N}'$*, then* $\mathtt{N} \equiv \mathtt{N}' \parallel \{l_1 \leftrightarrow l_2\}$*;*
(2) *if* $\mathtt{N} \xrightarrow{\langle l \rangle \,@\, l_1 : l_2} \mathtt{N}'$*, then* $\mathtt{N} \equiv \mathtt{N}' \parallel l_1 :: \langle l \rangle \parallel \{l_1 \leftrightarrow l_2\}$*;*
(3) *if* $\mathtt{N} \xrightarrow{(\nu l)\ \langle l \rangle \,@\, l_1 : l_2} \mathtt{N}'$*, then* $\mathtt{N} \equiv (\nu l)(\mathtt{N}' \parallel l_1 :: \langle l \rangle \parallel \{l_1 \leftrightarrow l_2\})$ *and* $l \notin \{l_1, l_2\}$.

Finally, Proposition 4.4 becomes as follows (in particular, cases 2., 3. and 6. are similar to the corresponding cases); we omit the proof because it is similar to that of Proposition 4.4.

Proposition 5.3 *Let* $\widetilde{l_1} \cap \widetilde{l_2} = \emptyset$*; then,* $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N} \parallel \mathtt{K}) \xrightarrow{\alpha} \bar{\mathtt{N}}$ *if and only if one of the following conditions holds, possibly exchanging* $\mathtt{K}$ *and* $\mathtt{N}$*:*

(1) $(\nu\widetilde{l_1})(\widetilde{l_2})\mathtt{N} \xrightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})\mathtt{N}'$ *and* $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{N}' \parallel \mathtt{K})$*. 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* $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{N}' \parallel \mathtt{K}')$*, where*
  - $\alpha = (\nu l)\ \langle l \rangle \,@\, l_1 : l_2$, $\widetilde{l'_1} = \widetilde{l_1} - \{l\}$ *and* $\widetilde{l'_2} = \widetilde{l_2} \cup \{l\}$*, if* $\widetilde{l} = \emptyset$ *and* $l \in \widetilde{l_1}$
  - $\alpha = (\nu\widetilde{l})\ \langle l \rangle \,@\, l_1 : l_2$, $\widetilde{l'_1} = \widetilde{l_1}$ *and* $\widetilde{l'_2} = \widetilde{l_2}$*, otherwise.*

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

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

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

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

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

Finally, we present a result 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 5.4 *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* $\mathtt{N}$*, 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 for the judgement $\mathtt{N} \xrightarrow{\alpha} \mathtt{N}'$. ∎

5.2 A Bisimulation-based Characterisation of Barbed Congruence

We can smoothly adapt Definition 4.5 to characterise barbed congruence also in the richer language. Notably, actions **conn** and **acpt** correspond quite closely to output/input prefixes of the synchronous π-calculus [26] and, thus, are handled similarly (see point 1. of Definition 5.5).

Definition 5.5 (Bisimilarity) *A symmetric relation* $\mathfrak{R}$ *between* TKLAIM *nets is a* bisimulation *if, for each* $N \,\mathfrak{R}\, M$ *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* $N' \,\mathfrak{R}\, M'$*, for some* M'*;*

(2) $\alpha = \exists ?\beta$ *implies that* $M \parallel \mathrm{NET}(\beta) \Rightarrow M'$ *and* $N' \,\mathfrak{R}\, M'$*, 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 5.6 (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 5.5. In the remainder of this section, we give full details of these key steps.

Lemma 5.7 $\approx$ *is context closed.*

Proof: We shall prove that the relation

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

is a bisimulation up-to $\equiv$; by taking $\widetilde{l} = \emptyset$, we obtain the thesis. Consider $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N} \parallel \mathtt{K}) \xrightarrow{\alpha} \bar{\mathtt{N}}$; by Proposition 5.3, we have sixteen cases. Indeed, since $\mathtt{K}$ is restriction and half-restriction free, cases 4(b) and 5(b) do not occur; moreover, there is no restriction in the label of the transition from $\mathtt{K}$ in cases 4(a) and 5(a) and in the symmetric of cases 2, 6 and 7. All the details are in Appendix 8. ∎

We now consider the completeness part that can be easily proved, as before, once we prove the following Lemma (that generalises Lemma 4.10).

Lemma 5.8

(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

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

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). It suffices to prove that the relation

$$\mathfrak{R} \triangleq \{(\,(\widetilde{l_1})N\ ,\ (\widetilde{l_1})M\,)\ :\ [\![N, \widetilde{l_1}, \widetilde{f_1}, \widetilde{l_2}, \widetilde{f_2}]\!] \cong [\![M, \widetilde{l_1}, \widetilde{f_1}, \widetilde{l_2}, \widetilde{f_2}]\!] \wedge (\widetilde{f_1}, \widetilde{f_2}) \neq \emptyset$$
$$(\widetilde{f_1}, \widetilde{f_2}) \cap fn(N, M, l) = \emptyset \wedge \widetilde{l_2} \subseteq fa(N) \cap fa(M)\}$$

is a bisimulation up-to $\equiv$. Indeed, by taking $\widetilde{l_1} = \{l\}$ and $\widetilde{l_2} = \emptyset$, we prove the first claim; vice versa, by taking $\widetilde{l_2} = \{l\}$ and $\widetilde{l_1} = \emptyset$, we prove the second claim. All the remaining details are in Appendix 8. ∎

Theorem 5.9 (Completeness) *If $N \cong M$ then $N \approx M$.*

Proof: We shall prove that the relation $\cong\ \cup\ \approx$ is a bisimulation up-to $\equiv$. Let $N \xrightarrow{\alpha} \mathtt{N}$ and reason by case analysis on α. Since the proof proceeds as in Theorem 4.11, 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 4.11, we get that $M \overset{\alpha}{\Longrightarrow} M'$ and, by Lemma 5.8(1), $(l')\,N' \approx (l')\,M'$; by (HEXT), (HGARB) and (PZERO), 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 4.11 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 and l_f be reserved and fresh names; now, consider the context

$$\mathcal{C}[\cdot] \triangleq (\nu l)\,([\cdot] \,\|\, l_1 :: \mathbf{acpt}(l_f) \,\|\, l_f :: \text{GO}\ l_1\ \text{DO}\ \mathbf{disc}(l_2)\ \text{THEN}\ (\mathbf{out}(l)@l_f \oplus \mathbf{nil}))$$

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 $\mathcal{C}[\cdot] \triangleq [\,\cdot\,] \,\|\, \text{NET}(\beta)$.

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

$$\mathcal{C}[\cdot] \triangleq (\nu l)\,([\cdot] \,\|\, l_1 :: \mathbf{acpt}(l_f) \,\|\, l_f :: \text{GO}\ l_1\ \text{DO}\ \mathbf{conn}(l_2).\mathbf{disc}(l_2)\ \text{THEN}$$
$$\mathbf{out}(l)@l_f \oplus \mathbf{nil})$$

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

$$\mathcal{C}[\cdot] \triangleq [\,\cdot\,] \,\|\, l_1 :: \mathbf{acpt}(l_f) \,\|\, l_f :: \text{GO}\ l_1\ \text{DO}\ \mathbf{acpt}(!x).\mathbf{eval}(\mathbf{acpt}(l_f))@x.\mathbf{disc}(x)$$
$$\text{THEN}\ \mathbf{conn}(x).(\mathbf{out}(x)@l_f \oplus \mathbf{nil})$$

where, with abuse of notation, we use process GO l DO _ THEN P introduced in Section 4.2 also when _ is a sequence of actions. 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_2 because they both have performed a transition labelled with $(\nu l_2)\ l_2 : ?l_1$. By Lemma 5.8(2), this implies that $N' \approx M'$, as required.

∎

6 Trace Equivalence for TKLAIM

In this section, we shall present a trace-based proof-technique for may testing. We start by recasting may testing $\simeq$ in a more standard formulation in terms of observers, computations and success of a computation [17], which justifies the name of the equivalence. 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.

Definition 6.1 (Observers) Observers*, ranged over by O, O', O_1, $\ldots$, 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 6.2 (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 $O_i \equiv O' \parallel \texttt{test} :: \langle\rangle$ and $\texttt{test} \notin \widetilde{l_i}$. We write N MAY O whenever there exists a successful computation from $N \parallel O$.*

Definition 6.3 *$N \simeq' M$ if, for every observer O, it holds that N MAY O if and only if M MAY O.*

Proposition 6.4 $\simeq \ = \ \simeq'$.

Proof: We start proving that $\simeq \subseteq \simeq'$. Let $N \simeq M$ and take an observer O such that N MAY O. Then, by context closure, $N \parallel O \ \simeq \ M \parallel O$ and, by barb preservation, $N \parallel O \Downarrow \texttt{test}$ (that comes from N MAY O) implies that $M \parallel O \Downarrow \texttt{test}$, i.e. M 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 $O \triangleq \texttt{test} :: \mathbf{in}(!x)@l.\mathbf{out}()@\texttt{test} \parallel \{\texttt{test} \leftrightarrow l\}$. Then, N MAY O that, by hypothesis, implies M 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 $\mathcal{C}[N]$ MAY O. We now consider the observer $H \parallel O$; by Definition 6.3, by induction and by the fact that $\mathcal{D}[N]$ MAY $H \parallel O$, we have that $\mathcal{D}[M]$ MAY $H \parallel O$. By rule (PASS) and because $\equiv \subseteq \simeq'$, this implies $\mathcal{C}[M]$ 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, $\mathcal{C}[N]$ MAY O if and only if $\mathcal{D}[N]$ MAY O (and similarly when replacing N with M). By induction, $\mathcal{D}[N] \simeq' \mathcal{D}[M]$; this suffices to conclude.

∎

For some well-known process calculi, may testing coincides with trace equivalence [17,5,6]. 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 (i.e., a sequence of actions different from τ), then M must exhibit σ as well, and vice versa. In an asynchronous setting [6,11], this requirement must be properly weakened because the discriminating power of asynchronous contexts is weaker: for example, the traditional formulation of trace equivalence would distinguish the 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.

By following the approach put forward in [6], a weaker trace-based equivalence can be defined by relying on a pre-order $\leq$ on traces (rather than using identity). Differently from [15], we discuss here only the adaption to TKLAIM of the laws borrowed from the asynchronous π-calculus [6]; indeed, in [15] we aimed at a complete characterisation of may testing in the simplified setting without the **acpt** primitive, whereas here the corresponding completeness result seems hard to obtain. The problem lies in the discriminating power of language contexts: to enable observations, it is usually necessary to make the observed nodes accepting connection requests originating from the context. This task can be easily accomplished for free addresses (by possibly exploiting process mobility), but not for restricted addresses; the latter ones cannot be forced to accept connection requests. A way to go around this problem and achieve completeness could be the definition of different operators than standard parallel composition to put together observers and observed nets. Such operators would enable the observation of restricted nodes and, thus, would make observers stronger than normal parallel components. We leave the formal development of this solution (and possibly different ones) for future research. However, it has to be said that using different operators would not respect the original approach of testing equivalences [17].

To define trace equivalence, we slightly modify the LTS of Section 5.1 by adding *bound input* and *bound acceptance* labels; they take the form $\exists ?(\nu l)\ \langle l \rangle @ l_1 : l_2$ and $l_1 : !(\nu l_2)\ l_2$, respectively, and are introduced to define a complementation function over labels:

$$\overline{(\nu\tilde{l})\ \langle l \rangle @ l_1 : l_2} \triangleq \exists ?(\nu\tilde{l})\ \langle l \rangle @ l_1 : l_2 \qquad \overline{\exists ?(\nu\tilde{l})\ \langle l \rangle @ l_1 : l_2} \triangleq (\nu\tilde{l})\ \langle l \rangle @ l_1 : l_2$$

$$\overline{l_1 \curvearrowright l_2} \triangleq \exists ? l_1 \curvearrowright l_2 \qquad \overline{\exists ? l_1 \curvearrowright l_2} \triangleq l_1 \curvearrowright l_2$$

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

Of course, $bn(\alpha)$ is extended by letting $bn(\exists ?(\nu l)\ \langle l \rangle @ l_1 : l_2) = bn(l' : !(\nu l)\, l) = \{l\}$, whereas $fn(\alpha)$ and $n(\alpha)$ are defined accordingly. Labels $\exists ?(\nu l)\ \langle l \rangle @ l_1 : l_2$ and $l' : !(\nu l)\, l$ are generated whenever a net N can perform $\exists ?\ \langle l \rangle @ l_1 : l_2$ and $l' : !l$, respectively, for $l \notin fn(N)$.

Notation 6.5 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 $N \overset{\phi}{\Longrightarrow}\overset{\sigma}{\Longrightarrow}$.

Then, we introduce a pre-order $\leq$ over traces. The intuition behind $\sigma' \leq \sigma$ 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 Lemma 6.8). Formally, $\leq$ is obtained as the least reflexive and transitive relation defined by the following laws, all inspired by [6]:

$$
\begin{array}{lrll}
\text{(L1)} & \sigma \cdot (\sigma')\backslash_{\tilde{l}} & \leq \sigma \cdot (\exists ?\beta \cdot \sigma')\backslash_{\tilde{l}} & \text{if } (\sigma')\backslash_{\tilde{l}} \neq \bot \\
\text{(L2)} & \sigma \cdot (\phi \cdot \exists ?\beta \cdot \sigma')\backslash_{\tilde{l}} & \leq \sigma \cdot (\exists ?\beta \cdot \phi \cdot \sigma')\backslash_{\tilde{l}} & \text{if } (\phi \cdot \exists ?\beta \cdot \sigma')\backslash_{\tilde{l}} \neq \bot \\
\text{(L3)} & \sigma \cdot (\sigma')\backslash_{\tilde{l}} & \leq \sigma \cdot (\exists ?\beta \cdot \beta \cdot \sigma')\backslash_{\tilde{l}} & \text{if } (\sigma')\backslash_{\tilde{l}} \neq \bot
\end{array}
$$

Intuitively, law (L1) states that labels representing 'requirements' cannot be directly observed. Law (L2) states that the execution of a 'requirement' action can be delayed along computations without being noticed by any observer. Finally, law (L3) states that two adjacent 'complementary' actions can be deleted. However, when moving/removing a label of the form $\exists ?(\nu l)\ \langle l \rangle @ l_1 : l_2$, we must keep the information that l is a fresh received name. To this aim, we exploit the function $(\sigma)\backslash_{\tilde{l}}$ whose formal definition is:

$$
(\sigma)\backslash_{\tilde{l}} \triangleq \sigma \qquad \text{if } \tilde{l} \cap fn(\sigma) = \emptyset
$$

$$
(\phi \cdot \sigma)\backslash_{l} \triangleq \begin{cases} \exists ?(\nu l)\ \langle l \rangle @ l_1 : l_2\ \cdot \sigma & \text{if } \phi = \exists ?\ \langle l \rangle @ l_2 : l_1 \text{ and } l \notin \{l_1, l_2\} \\ \phi \cdot (\sigma)\backslash_{l} & \text{if } l \notin n(\phi) \text{ and } (\sigma)\backslash_{l} \neq \bot \\ \bot & \text{otherwise} \end{cases}
$$

To better understand the motivations underlying this definition, consider the following example that justifies the side condition of law (L1) (similar arguments also hold for laws (L2) and (L3)). In the trace $\exists ?(\nu l)\ \langle l \rangle @ l_2 : l_1 \ \cdot\ \langle l \rangle @ l_3 : l_4$ performed by N, the input action cannot be erased. Indeed, since l is fresh, 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 the trace $\exists ?(\nu l)\ \langle l \rangle @ l_6 : l_5 \ \cdot\ \langle l \rangle @ l_3 : l_4$ is smaller (w.r.t. $\leq$) than $\exists ?(\nu l)\ \langle l \rangle @ l_2 : l_1 \ \cdot \exists ?\ \langle l \rangle @ l_6 : l_5 \ \cdot\ \langle l \rangle @ l_3 : l_4$.

Finally, we are ready to define trace equivalence.

Definition 6.6 (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 $\sigma' \leq \sigma$.*

We now prove that $\asymp$ is a sound proof-technique for $\simeq$. More precisely, we shall prove that trace equivalent nets are also may testing equivalent, in the sense of Definition 6.3; because of Proposition 6.4 this suffices to conclude. Thus, we use observers O and denote OK the label $\langle\rangle$ @ `test` : `test`, i.e., the action that must be exhibited in any successful computation.

To prove soundness, we need two auxiliary lemmata. The first one states that a net can report success when run in parallel with an observer if and only if they execute complementary traces (where the complementation function is pointwise extended to sequences of actions as expected). The second one states that the laws defining $\leq$ are 'sound', in the sense that, whenever $\sigma' \leq \sigma$ and $N \overset{\sigma'}{\Longrightarrow}$, any observer able to perform $\overline{\sigma}$ may report success when run in parallel with N.

Lemma 6.7 *$N \parallel O \overset{\mathtt{OK}}{\Longrightarrow}$ if and only if $N \overset{\sigma}{\Longrightarrow}$ and $O \overset{\overline{\sigma}\cdot\mathtt{OK}}{\Longrightarrow}$.*

Proof: The "if" part is proved by induction on the length of σ. For the "only if" part, it must be that $N \parallel O\ (\overset{\tau}{\longrightarrow})^n H \overset{\mathtt{OK}}{\longrightarrow}$; the proof is by induction on n and exploits Proposition 5.3 to examine all the possible interactions among N and O. ∎

Lemma 6.8 *Let $\sigma' \leq \sigma$, $N \overset{\sigma'}{\Longrightarrow}$ and $O \overset{\overline{\sigma}\cdot\mathtt{OK}}{\Longrightarrow}$; then, $N \parallel O \overset{\mathtt{OK}}{\Longrightarrow}$.*

Proof: By induction on the number of laws used to derive $\sigma' \leq \sigma$, we prove that $O \overset{\overline{\sigma'}\cdot\mathtt{OK}}{\Longrightarrow}$; by Lemma 6.7, this suffices to conclude. ∎

Theorem 6.9 (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 6.7, there exists σ such that $N \overset{\sigma}{\Longrightarrow}$ and $O \overset{\overline{\sigma}\cdot\mathtt{OK}}{\Longrightarrow}$. By Definition 6.6, there exists $\sigma' \leq \sigma$ such that $M \overset{\sigma'}{\Longrightarrow}$; by Lemma 6.8, it holds that $M \parallel O \overset{\mathtt{OK}}{\Longrightarrow}$, as required by Definition 6.3. ∎

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 verify a relevant property. The scenario we consider is inspired by the *handover protocol*, proposed by the European Telecommunication Standards Institute 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 [29].

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 exits from the area associated to the BS and enters 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]. 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 fields and corresponding fields 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 used to replace 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., [21]).

In our implementation, MSs, BSs and MSCs are modelled as nodes. We consider a simple PLMN, with one MSC (whose address is *msc*), n BSs (whose addresses are $bs_1, \dots, bs_n$, 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 handovers). The handover for l is handled by the MSC via the following process:

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

where $\mathbf{eval}_u(P)$ is a more readable way of writing process $\mathbf{eval}(P)@u$; similarly, $\mathbf{eval}_{u,v}(P)$ stands for $\mathbf{eval}(\mathbf{eval}(P)@v)@u$. Process *HNDVR* first 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

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

The main property we want to ensure in this scenario is that the MS at 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:

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

Soundness of the protocol can be established by proving that SYS'_i is behaviourally equivalent to $SPEC$, where

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

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. Thus, $SPEC$ can be seen as a specification of the desired behaviour of any implementation of the system.

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:

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

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 $SPEC$:

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

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

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

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

$$
\begin{aligned}
M^0_{i,j,k} &\triangleq (\nu\widetilde{l})\,(PLMN \;\|\; msc :: \mathbf{in}(!y)@table.\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.\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{eval}_l(\mathbf{acpt}(bs_j)).\cdots) \;\|\; \{bs_i \leftrightarrow l\}) \\
&\qquad \|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^k \\
M^3_{i,j,k} &\triangleq (\nu\widetilde{l})\,(PLMN \;\|\; bs_i :: \mathbf{eval}_l(\mathbf{acpt}(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{disc}(l).\cdots \;\|\; \{bs_i \leftrightarrow l\} \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^k \mid \mathbf{acpt}(bs_j)) \\
M^5_{i,j,k} &\triangleq (\nu\widetilde{l})\,(PLMN \;\|\; bs_i :: \mathbf{eval}_{msc,bs_j}(\mathbf{conn}(l).\cdots) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^k \mid \mathbf{acpt}(bs_j)) \\
M^6_{i,j,k} &\triangleq (\nu\widetilde{l})\,(PLMN \;\|\; msc :: \mathbf{eval}_{bs_j}(\mathbf{conn}(l).\cdots) \;\|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^k \mid \mathbf{acpt}(bs_j)) \\
M^7_{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 \mid \mathbf{acpt}(bs_j)) \\
M^8_{i,j,k} &\triangleq (\nu\widetilde{l})\,(PLMN \;\|\; bs_j :: \mathbf{eval}_{msc}(\,\mathbf{out}(l, bs_j)@table\,) \;\|\; \{bs_j \leftrightarrow l\}) \\
&\qquad \|\; l :: (\langle\text{“}conn\text{”}, l, msc\rangle)^k \\
M^9_{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{aligned}
$$

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

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

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

$$\mathfrak{R} \triangleq \bigcup_{\substack{k\geq 0\\ i=1..n}} \{(SYS'_{i,k}\ ,\ SPEC_k)\} \ \cup \bigcup_{\substack{k\geq 0\\ i=1..n\\ h=0..3}} \{(N^{h}_{i,k}\ ,\ SPEC_k)\} \ \cup \bigcup_{\substack{k\geq 0\\ i,j\in\{1..n\}\\ h=0..9}} \{(M^{h}_{i,j,k}\ ,\ SPEC_k)\}$$

Indeed, $\mathfrak{R}$ contains $(SYS'_i, SPEC)$ up-to $\equiv$, because $SYS'_i \equiv SYS'_{i,0}$ and $SPEC \equiv SPEC_0$.

We now prove that $\mathfrak{R}$ is 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 $M^h_{i,j,k} \Rightarrow SYS'_{j,k} \Rightarrow N^2_{j,k+1}$.

A trace-based proof of equivalence 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 $\sigma' \leq \sigma$. 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), $SYS'_{i,k} \xRightarrow{msc \curvearrowright msc} SYS'_{i,k'}$ and, by induction, there exists a $\sigma_2 \leq \sigma_1$ 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 before.

$\exists?l \curvearrowright l$**:** 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 $\sigma_2 \leq \sigma_1$ 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 $\sigma' \triangleq \sigma_2 \leq \sigma_1 \leq \phi \cdot \sigma_1 \triangleq \sigma$.

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. $SYS'_{i,k} \Rightarrow K \xrightarrow{\phi} K' \overset{\sigma'}{\Longrightarrow}$. If $K' \triangleq SYS'_{i',k'}$, for some i' and k', the thesis follows by an easy induction. Otherwise, we have two possible sub-cases:

$K' \triangleq M^h_{i',j,k'}$**:** to apply induction, we need to let K' produce σ' through a net of the form $SYS'_{i',k'}$. Therefore, we consider the following alternative way[3] 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, $SPEC_k \Rightarrow SPEC_{k''} \xrightarrow{\phi} SPEC_{k'}$, for $k'' = k'$, if transitions (17) or (18) have been used to derive K' from K, and $k'' = k' + 1$, if (19) has been used. By induction, $SPEC_{k'} \overset{\sigma'}{\Longrightarrow}$; this suffices to conclude.

$K' \triangleq N^h_{i',k'}$: consider $K' \Rightarrow SYS'_{i',k''} \Rightarrow N^h_{i',k'+1} \overset{\sigma'}{\Longrightarrow}$, where $k'' = k'$, if $h = 2, 3$, and $k'' = k' + 1$, if $h = 0, 1$. Again, take $SPEC_k \Rightarrow SPEC_{k'''} \xrightarrow{\phi} SPEC_{k'+1}$, for $k''' = k' + 1$, if transitions (11) or (12) have been used to derive K' from K, and $k''' = k' + 2$, if (13) has been used. By induction, $SPEC_{k'+1} \overset{\sigma'}{\Longrightarrow}$; this suffices to conclude.

8 Conclusions and Related Work

We have presented operational and observational semantics for tKlaim, a process calculus equipped with primitives for process distribution and mobility, remote and asynchronous communication through distributed data repositories, and dynamic activation/deactivation of inter-node connections. The semantic theories we introduced in this paper have been defined in a uniform fashion [7]: first, we defined some basic observables for a global computing setting; then, we closed them under all possible contexts and/or reductions, to obtain *barbed congruence* and *may testing*; finally, we gave a more tractable characterisation of the former equivalence by means of a *labelled bisimulation*.

The language proposed and its semantic theories have proved suitable to program and verify a non-trivial example, inspired by the *handover protocol*. This example shows that working with bisimilarity is easier than working with trace equivalence. To establish bisimilarity, we had only to find, for every action of one net, a proper reply of the other net. To establish trace equivalence, a more sophisticated inductive reasoning was needed.

We believe that, although tKlaim can be somehow encoded in the π-calculus, the introduction of the former is justified by at least two reasons.

(1) tKlaim permits a direct description of key features of global computers such as process distribution and mobility, and inter-node connections. An encoding of such features in π-calculus although possible would hide them within complex process structures, making it difficult to reason on the resulting processes. tKlaim and π-calculus can be seen as formalisms standing at two different

[3] 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.

levels of abstraction: TKLAIM enables the user to exploit the knowledge of the topology of the net; the π-calculus (and especially its dialects more suitable for distributed implementations, like Join calculus [18] and $\pi_{1\ell}$-calculus [1]) permits directly referring to network sockets (that can be represented as communication channels).

(2) A convincing encoding should enjoy ‘reasonable’ properties, like those pointed out in [30]. We believe this cannot be the case for the encoding of TKLAIM. For example, in [14] we presented the encoding of a TKLAIM’s sub-calculus into the asynchronous π-calculus and argued that it is not possible to devise a convergence preserving encoding. We are now working on proving that, due to the check of existence of the target of a communication that is performed in TKLAIM and not in the π-calculus, a divergence free encoding does not exist.

We conclude by touching upon most strictly related work and some possible developments.

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. Here, we want to mention work on equivalences for such languages.

Bisimulation-based characterisations of barbed congruences for calculi relying on a flat net topology are developed in [1,22]; such characterisations are mainly derived from bisimulation equivalences for the π-calculus and its variants. Bisimulation-based characterisations of barbed congruences for calculi relying on a hierarchical net topology are developed in [25,8,10,23]. All such bisimulations follow Sangiorgi’s *context bisimulation* [33], thus they still rely on a universal quantification over processes. Moreover, the bisimulations introduced in [10,23] are not complete proof techniques for the corresponding barbed congruences.

Even though processes can occur in TKLAIM as arguments of actions **eval**, our formalism shares with calculi relying on a flat net topology the fact that the LTS and the associated bisimulation do not use labels containing processes. Indeed, the bisimulation relies only on a standard quantification over names (when considering labels of the form $\exists ?\ \langle l \rangle\ @\ l_2 : l_1$) and we strongly conjecture that it is decidable, for non-trivial fragments of the language: techniques similar to those in [28] could be used here. But we have also that TKLAIM shares with calculi relying on a hierarchical net topology the complexity of the underlying LTS. As a partial defence we can say that the LTS is just an adequate tool to establish the alternative characterisation of barbed congruence and it is not intended as a tool to define the operational semantics of TKLAIM.

The work most closely related to ours is [19]. There, a distributed version of the π-calculus is presented where nodes are connected through links that can fail during the computation; a bisimulation-based proof technique is used to establish properties of systems. A notable difference is that in TKLAIM a deactivated connection can be re-established later on, via the primitives **conn/acpt**, whereas this is not possible in $D\pi_F$. Thus we have that in $D\pi_F$ link failures are permanent, whereas in TKLAIM they can also be transient.

Future work It would be worth studying forms 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 [22].

Orthogonally, it would also be interesting to analyse efficiency issues to better clarify, e.g., the advantages of mobile code and process distribution. A possible application is to study possible rearrangements of the processes over a given net to minimise the number of remote operations, that are normally more expensive and slower than local ones. A simple way to model this scenario is to assign costs to connections (see, e.g., [12]) and develop efficiency preorders based on such information.

Appendix A: Technical Proofs

Proof of Proposition 4.4: Let $(\nu\widetilde{l})(N \parallel K) \xrightarrow{\alpha} \bar{N}$; we reason by induction on the depth of the shortest inference for $\xrightarrow{\alpha}$ and prove that only one of the cases enumerated in this Proposition is possible. We have three base cases (of depth 2); in all of them, $\widetilde{l} = \emptyset$ and the hypotheses are axioms from Table 4. We analyse the last rule used in the inference:

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

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, $(\nu\widetilde{l'})(N \parallel K) \xrightarrow{\alpha} \bar{N}'$ and $\bar{N} \triangleq (\nu l)\bar{N}'$, for $l \notin n(\alpha)$. By induction on $(\nu\widetilde{l'})(N \parallel K) \xrightarrow{\alpha} \bar{N}'$, that has a shorter inference, we fall in one of the cases of this Proposition. In the same case falls also the inference for $(\nu\widetilde{l})(N \parallel K) \xrightarrow{\alpha} \bar{N}$.

(LTS-OPEN): now $\widetilde{l} = (l, \widetilde{l'})$, $(\nu\widetilde{l'})\,(N \parallel K) \xrightarrow{\langle l\rangle\, @\, l_1 : l_2} \bar{N}'$, $\alpha = (\nu l)\ \langle l\rangle\ @\ l_1\ :\ l_2$ and $\bar{N} \triangleq \bar{N}'$. Thus, we apply induction and we can only fall in cases 1 or 2 (or their symmetric versions); in the same case falls the inference from $(\nu\widetilde{l})\,(N \parallel K)$.

(LTS-STRUCT): we reason by case analysis on the axiom of Table 2 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 Proposition.

(PCOM): $\widetilde{l} = \emptyset$ and $K \parallel N \xrightarrow{\alpha} \bar{N}'$, for $\bar{N}' \equiv \bar{N}$. By induction on $K \parallel N \xrightarrow{\alpha} \bar{N}'$, that has a shorter inference, we fall in one of the cases of this Proposition. 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 $N_1 \parallel (N_2 \parallel K) \xrightarrow{\alpha} \bar{N}'$, for $\bar{N}' \equiv \bar{N}$. We now apply induction and reason on the case in which the latter transition falls:

Case 1: then, $N_1 \xrightarrow{\alpha} N_1'$ and $\bar{N}' \triangleq N_1' \parallel (N_2 \parallel K)$. We still easily fall in case 1.

Symmetric of case 1: $N_2 \parallel K \xrightarrow{\alpha} \bar{N}''$ and $\bar{N}' \triangleq N_1 \parallel \bar{N}''$: we apply induction to $N_2 \parallel K \xrightarrow{\alpha} \bar{N}''$; the case for $(N_1 \parallel N_2) \parallel K$ is the same as that obtained in this latter inductive step.

Case 2: then, $N_1 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle\, @\, l_1 : l_1} N_1'$, $N_2 \parallel K \xrightarrow{l_1 \curvearrowright l_2} \bar{N}''$ and $\bar{N}' \triangleq N_1' \parallel \bar{N}''$. 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.

Cases 3, 5, 6, and symmetric of cases 2, 3 and 4: similar to the previous case.

Case 4: then, $N_1 \xrightarrow{\exists?\ \langle l\rangle\, @\, l_1 : l_2} N_1'$, $N_2 \parallel K \xrightarrow{(\nu l)\ \langle l\rangle\, @\, l_2 : l_1} \bar{N}''$ and $\bar{N}' \triangleq (\nu l)\,(N_1' \parallel \bar{N}'')$: 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.

Symmetric of case 5: $N_1 \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle\, @\, l_2 : l_2} N_1'$, $N_2 \parallel K \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{\exists?\ \langle l\rangle\, @\, l_2 : l_1} \bar{N}''$ and $\bar{N}' \triangleq (\nu\widetilde{l'})\,(N_1' \parallel \bar{N}'')$: 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.

Symmetric of case 6: $N_1 \xrightarrow{l_2 \curvearrowright l_1} N_1'$, $N_2 \parallel K \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle @ l_2 : l_2} \xrightarrow{\exists?\ \langle l\rangle @ l_2 : l_1} \bar{N}''$ and $\bar{N}' \triangleq (\nu\widetilde{l'})(N_1' \parallel \bar{N}'')$: 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 $(N \parallel K_1) \parallel K_2 \xrightarrow{\alpha} \bar{N}'$.

(RCOM)**:** $\widetilde{l} = (l_1, l_2, \widetilde{l'})$ and $(\nu l_2)(\nu l_1)(\nu\widetilde{l'})(N \parallel K) \xrightarrow{\alpha} \bar{N}'$, for $\bar{N}' \equiv \bar{N}$; 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, $K \triangleq (\nu l)\bar{K}$ and $(\nu l)(N \parallel \bar{K}) \xrightarrow{\alpha} \bar{N}'$, for $\bar{N}' \equiv \bar{N}$ and $l \notin fn(N)$. Now, we can apply induction to $(\nu l)(N \parallel \bar{K}) \xrightarrow{\alpha} \bar{N}'$ and conclude that $N \parallel K$ evolves in the same way as $(\nu l)(N \parallel \bar{K})$; just notice that, whenever $\bar{K} \xrightarrow{\langle l\rangle @ l_2 : l_1} K'$ 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, whenever $(\nu l) K \xrightarrow{(\nu l)\ \langle l\rangle @ l_2 : l_1} K'$ arises, it will be replaced by $K \xrightarrow{\langle l\rangle @ l_2 : l_1} K'$.

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

Proof of Lemma 5.7: Let $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N} \parallel \mathtt{K}) \xrightarrow{\alpha} \bar{\mathtt{N}}$; by Proposition 5.3, we have sixteen possible interactions.

(1) *(Proposition 5.3(1))* $(\nu\widetilde{l_1})(\widetilde{l_2})\mathtt{N} \xrightarrow{\alpha} (\nu\widetilde{l_1'})(\widetilde{l_2'})\mathtt{N}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1'})(\widetilde{l_2'})(\mathtt{N}' \parallel \mathtt{K})$; 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, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \overset{\hat{\alpha}}{\Longrightarrow} (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \bar{\mathtt{M}}$ and, by definition, $\bar{\mathtt{N}}\ \mathfrak{R}\ \bar{\mathtt{M}}$.

(ii) $\alpha = \exists?\beta$. By (EXT), we have that $(\widetilde{l})\mathtt{M} \parallel \mathrm{NET}(\beta) \equiv (\widetilde{l})(\mathtt{M} \parallel \mathrm{NET}(\beta)) \Rightarrow (\widetilde{l})\mathtt{M}'$ and $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$; hence, by (EXT) and (HEXT), it holds that $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \parallel \text{NET}(\beta) \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \text{NET}(\beta) \parallel \mathtt{K}) \Rightarrow (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \bar{\mathtt{M}}$ and $\bar{\mathtt{N}} \,\mathfrak{R}\, \bar{\mathtt{M}}$.

(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, $(\widetilde{l})\mathtt{M} \xRightarrow{\alpha'} (\widetilde{l})\mathtt{M}'$ and $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$. Then, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \xRightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \bar{\mathtt{M}}$ and $\bar{\mathtt{N}} \,\mathfrak{R}\, \bar{\mathtt{M}}$, because $\widetilde{l}$ 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 similar to case 1(a).i, whereas the case for $\alpha = \exists?\, \langle l\rangle @ l_1 : l_2$ is similar to case 1(a).ii.

(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, $(\widetilde{l})\mathtt{N} \xRightarrow{\alpha'} (\widetilde{l'})\mathtt{N}'$, for $\widetilde{l'} = \widetilde{l} - \{l\}$; thus, $(\widetilde{l})\mathtt{M} \xRightarrow{\alpha'} (\widetilde{l'})\mathtt{M}'$ and $(\widetilde{l'})\mathtt{N}' \approx (\widetilde{l'})\mathtt{M}'$. Then, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \xRightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}) \triangleq \bar{\mathtt{M}}$ and $\bar{\mathtt{N}} \,\mathfrak{R}\, \bar{\mathtt{M}}$, because $\widetilde{l'} \subseteq \widetilde{l}$ 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 5.3(1))* $(\nu\widetilde{l_1})(\widetilde{l_2})\mathtt{K} \xrightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})\mathtt{K}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{N} \parallel \mathtt{K}')$; 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. We only give details for point (a); points (b)/.../(e) can be proved by arguments easy derivable from what follows. 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, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \xrightarrow{\tau} (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{M} \parallel \mathtt{K}') \triangleq \bar{\mathtt{M}}$; by definition, $\bar{\mathtt{N}} \,\mathfrak{R}\, \bar{\mathtt{M}}$, because $\widetilde{l} \subseteq (\widetilde{l'_1}, \widetilde{l'_2})$ and $fa(\mathtt{K}') = fa(\mathtt{K})$ (see Proposition 5.4).

(ii) $\alpha = \langle l\rangle @ l_1 : l_2$. By Proposition 5.4, $fa(\mathtt{K}') = fa(\mathtt{K})$. Moreover, by definition of the LTS, it must be $\{l_1, l_2\} \cap (\widetilde{l_1}, \widetilde{l_2}) = \emptyset$. If $l \in \widetilde{l_1}$, then $\widetilde{l'_1} \triangleq \widetilde{l_1} - \{l\}$ and $\widetilde{l'_2} = \widetilde{l_2} \cup \{l\}$; otherwise, $\widetilde{l'_1} = \widetilde{l_1}$ and $\widetilde{l'_2} = \widetilde{l_2}$. In both cases, we conclude as before, because $\widetilde{l}$ is still a subset of $(\widetilde{l'_1}, \widetilde{l'_2})$.

(iii) $\alpha \in \{l_1 \curvearrowright l_2,\ l_1 : ?l_2,\ l_1 : !l_2 \}$. 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 $l_1 \in \widetilde{l_1}$ (this case is possible only for $\alpha = l_1 : ?l_2$), 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, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \xrightarrow{\alpha} (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{M} \parallel \mathtt{K}') \triangleq \bar{\mathtt{M}}$ and, by definition, $\bar{\mathtt{N}} \,\mathfrak{R}\, \bar{\mathtt{M}}$, because $\widetilde{l}$ is still a subset of $(\widetilde{l'_1}, \widetilde{l'_2})$ and $\widetilde{l} \cap fa(\mathtt{K}') = \emptyset$.

(iv) $\alpha = \exists?\beta$. Then $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \parallel \text{NET}(\beta) \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K} \parallel \text{NET}(\beta)) \xrightarrow{\tau} (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}') \triangleq \bar{\mathtt{M}}$ and, trivially, $\bar{\mathtt{N}} \,\mathfrak{R}\, \bar{\mathtt{M}}$. Indeed, by Proposition 5.4, $fa(\mathtt{K}') = fa(\mathtt{K}) \cup \{l_2\}$ but, by definition of the LTS, $l_2 \notin (\widetilde{l_1}, \widetilde{l_2})$; thus, $fa(\mathtt{K}') \cap \widetilde{l} = \emptyset$.

(3) *(Proposition 5.3(2))* $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle\,@\,l_2:l_2} (\widetilde{l'})\mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{K}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'_1})(\widetilde{l'_2})(\mathtt{N}' \parallel \mathtt{K}')$; if $\widetilde{l'_1} = \widetilde{l_1}$ and $\widetilde{l'_2} = \widetilde{l_2}$, the proof is derivable from case 13; if $\widetilde{l'_1} = \widetilde{l_1} - \{l\}$ and $\widetilde{l'_2} = \widetilde{l_2} \cup \{l\}$, the proof is similar to case 1(b).

(4) *(symmetric of Proposition 5.3(2))* $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{\langle l\rangle\,@\,l_2:l_2} \mathtt{K}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; this case is easy derivable from case 14.

(5) *(Proposition 5.3(3))* $\mathtt{N} \xrightarrow{\exists? l_1 \curvearrowright l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{K}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; 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 5.2(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 5.3(3))* $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{N}'$, $\mathtt{K} \xrightarrow{\exists? l_1 \curvearrowright l_2} \mathtt{K}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; then, since $l_1 \in fa(\mathtt{K})$, we have that $l_1 \notin \widetilde{l}$. If $l_2 \notin \widetilde{l}$, then $(\widetilde{l})\mathtt{M} \xRightarrow{l_1 \curvearrowright l_2} (\widetilde{l})\mathtt{M}'$ 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 $\mathtt{N} \xrightarrow{l_2 \curvearrowright l_1} \mathtt{N}'$ (that must hold whenever $\mathtt{N} \xrightarrow{l_1 \curvearrowright l_2} \mathtt{N}'$ holds). Thus, $(\widetilde{l})\mathtt{N} \xrightarrow{l_2 \curvearrowright l_1} (\widetilde{l'})\mathtt{N}'$, for $\widetilde{l'} = \widetilde{l} - \{l_2\}$; then, $(\widetilde{l})\mathtt{M} \xRightarrow{l_2 \curvearrowright l_1} (\widetilde{l'})\mathtt{M}'$ and $(\widetilde{l'})\mathtt{N}' \approx (\widetilde{l'})\mathtt{M}'$; this suffices to conclude, as $fa(\mathtt{K}') \cap \widetilde{l'} = \emptyset$.

(7) *(Proposition 5.3(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 $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; this case is similar to case 5.

(8) *(symmetric of Proposition 5.3(4).a)* $\mathtt{K} \xrightarrow{\exists?\ \langle l\rangle\,@\,l_2:l_1} \mathtt{K}'$, $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle\,@\,l_2:l_1} (\widetilde{l'})\mathtt{N}'$, $\widetilde{l'} \cap fn(\mathtt{K}) = \emptyset$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'}, \widetilde{l_1})\,(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$: again, by definition of the LTS, $l_1 \in fa(\mathtt{K})$. So, $l_1 \notin \widetilde{l}$, whereas it may be either $l_2 \in \widetilde{l}$ or not; we only consider the first case, as the second one is simpler. Then, $(\widetilde{l})\mathtt{N} \xrightarrow{l_2 \curvearrowright l_1} \xrightarrow{(\nu\widetilde{l'})\ \langle l\rangle\,@\,l_2:l_2} (\widetilde{l''})\mathtt{N}'$, 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 5.2, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \Rightarrow (\nu\widetilde{l'}, \widetilde{l_1})\,(\widetilde{l_2})(\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \parallel \mathtt{K}) \xrightarrow{\tau} (\nu\widetilde{l'}, \widetilde{l_1})\,(\widetilde{l_2})(\mathtt{M}' \parallel \mathtt{K}') \triangleq \bar{\mathtt{M}}$ and $\bar{\mathtt{N}}\ \mathfrak{R}\ \bar{\mathtt{M}}$: indeed, $\widetilde{l''} \subseteq (\widetilde{l'}, \widetilde{l_1}, \widetilde{l_2})$ and, because $\widetilde{l'} \cap fn(\mathtt{K}) = \emptyset$, it holds that $\widetilde{l''} \cap fa(\mathtt{K}') = \emptyset$.

(9) *(symmetric of Proposition 5.3(4).b)* $\mathtt{K} \xrightarrow{\exists?\ \langle l\rangle\,@\,l_2:l_1} \mathtt{K}'$, $\mathtt{N} \equiv (l)\,\mathtt{N}'$, $\mathtt{N}' \xrightarrow{\langle l\rangle\,@\,l_2:l_1} \mathtt{N}''$, $l \notin fa(\mathtt{K})$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})\,(l, \widetilde{l_2})\,(\mathtt{N}'' \parallel \mathtt{K}')$: this case can be proved like case 8; notice that here we have $\widetilde{l''} = (\widetilde{l} - \{l_2\})$.

(10) *(Proposition 5.3(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 $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}'' \parallel \mathtt{K}')$; 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 $\widetilde{l'} = \widetilde{l} - \{l_1\}$; 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 5.2, $(\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M} \parallel \mathtt{K}) \Rightarrow (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \parallel \mathtt{K}') \Rightarrow (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{M}'' \parallel \mathtt{K}') \triangleq \bar{\mathtt{M}}$ and $\bar{\mathtt{N}}\ \mathfrak{R}\ \bar{\mathtt{M}}$.

(11) *(symmetric of Proposition 5.3(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 $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'}, \widetilde{l_1})\,(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$: this is similar to case 6 but simpler, because $\widetilde{l} \cap \{l_1, l_2\} = \emptyset$.

(12) *(symmetric of Proposition 5.3(5).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 $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})\,(l,\widetilde{l_2})\,(\mathtt{N}'' \parallel \mathtt{K}')$: this case is similar to case 9 but simpler, because $\widetilde{l} \cap \{l_1, l_2\} = \emptyset$.

(13) *(Proposition 5.3(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 $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'},\widetilde{l_1})\,(\widetilde{l_2})(\mathtt{N}'' \parallel \mathtt{K}')$; by definition of $\mathtt{K}$, it holds that $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$. Hence, $(\widetilde{l})\mathtt{M} \xRightarrow{(\nu\widetilde{l'})\ \langle l\rangle\,@\,l_2:l_2} (\widetilde{l},\widetilde{l'})\,\mathtt{M}'$, $(\widetilde{l},\widetilde{l'})\,\underline{\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\}} \parallel l_2 :: \langle l\rangle \equiv (\widetilde{l},\widetilde{l'})\,(\mathtt{M}' \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \underline{\langle l\rangle}) \Rightarrow (\widetilde{l},\widetilde{l'})\,\mathtt{M}''$ 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.

(14) *(symmetric of Proposition 5.3(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 $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}')$; notice that $\{l_1, l_2\} \cap \widetilde{l} = \emptyset$ and easily conclude.

(15) *(Proposition 5.3(7))* $\mathtt{N} \xrightarrow{(\nu\widetilde{l'})\ l_2:?l_1} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1:!l_2} \mathtt{K}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l'},\widetilde{l_1})\,(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}' \parallel \{l_1 \leftrightarrow l_2\})$; 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 $l_2 \notin \widetilde{l}$; so, $(\widetilde{l})\mathtt{M} \xRightarrow{(\nu l_2)\ l_2:?l_1} (\widetilde{l})\mathtt{M}'$, for $(\widetilde{l})\mathtt{N}' \approx (\widetilde{l})\mathtt{M}'$, and we easily conclude. If $\widetilde{l'} = \emptyset$, we reason like in case 6.

(16) *(symmetric of Proposition 5.3(7))* $\mathtt{N} \xrightarrow{l_2:!l_1} \mathtt{N}'$, $\mathtt{K} \xrightarrow{l_1:?l_2} \mathtt{K}'$ and $\bar{\mathtt{N}} \equiv (\nu\widetilde{l_1})(\widetilde{l_2})(\mathtt{N}' \parallel \mathtt{K}' \parallel \{l_1 \leftrightarrow l_2\})$; similar to case 15. ∎

Proof of Lemma 5.8: 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 5.4), nodes in $\widetilde{l_2}$ will remain present in any reduct of N and M. We reason by case analysis on α;[4] moreover, to lighten notations, 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 $[\![N]\!]$.

(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, $\mathtt{N}\ \mathfrak{R}\ (\widetilde{l_1})M'$ up-to $\equiv$.

(2) $\alpha = l \curvearrowright l'$**:** by definition of the LTS, $l' \notin \widetilde{l_1}$; however, l' can belong to $\widetilde{l_2}$ or not, whereas 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 $l_f \in (\widetilde{f_1},\widetilde{f_2})$ and consider the context

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

[4] We follow here a way of reasoning similar to the one used in Theorem 4.11. Thus, we shall only give the discriminating context $C[\cdot]$ and some details on the key issues.

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

(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

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

Like before, we consider the reductions $\mathcal{C}[\llbracket N \rrbracket] \Longmapsto \llbracket N', \widetilde{l'_1}, \widetilde{f_*}, \widetilde{l'_2}, \widetilde{f'_*} \rrbracket \parallel f_i :: \mathbf{nil}$, where $\widetilde{f_*} \triangleq \widetilde{f_1} - \{f_i\}$, $\widetilde{l'_2} \triangleq \widetilde{l_2} \cup \{l_i\}$ and $\widetilde{f'_*} \triangleq \widetilde{f_2} \cup \{f'_{h+1}\}$. Because of freshness of f_i, $\llbracket N', \widetilde{l'_1}, \widetilde{f_*}, \widetilde{l'_2}, \widetilde{f'_*} \rrbracket \parallel f_i :: \mathbf{nil} \cong \llbracket N', \widetilde{l'_1}, \widetilde{f_*}, \widetilde{l'_2}, \widetilde{f'_*} \rrbracket$; we can easily conclude.

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

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

(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)

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

(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

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

(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) $\alpha = \langle l \rangle @ l' : l'$ **:** this case is similar to case 2., with action $\mathbf{in}(\cdot)@l'$ replacing $\mathbf{disc}(\cdot)$ in $\mathcal{C}[\cdot]$.

(4) $\alpha = (\nu l)\ \langle l \rangle @ l' : l'$ **:** we have two sub-cases:

(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

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

Similarly to the 3rd case in the proof of Theorem 4.11, 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

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

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

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

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

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

(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) $\alpha = l : !l'$ **:** this case is similar to case 2., with actions $\mathbf{conn}(\cdot).\mathbf{disc}(\cdot)$ replacing $\mathbf{disc}(\cdot)$.

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

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

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

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

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

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

(9) $\alpha = \exists?\langle l\rangle @ l'' : l'$ **:** by definition of the LTS, $\{l', l''\} \cap \widetilde{l_1} = \emptyset$, whereas it can be $l \in \widetilde{l_1}$; moreover, l, l' and l'' can belong to $\widetilde{l_2}$ or not. By also distinguishing whether $l = l'$, $l = l''$ and $l' = l''$, we have nineteen sub-cases:

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

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

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

(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

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

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

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

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

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

(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'' everywhere, except for $\mathbf{out}(y)@l''$ that becomes $\mathbf{out}(y)@x$.

(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'' everywhere, except for $\mathbf{out}(y)@l''$ that becomes $\mathbf{out}(y)@x$.

(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

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

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

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

(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):

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

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

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

(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 f'_j in place of f_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. ∎

References

[1] R. Amadio. On modelling mobility. *Theor. Comp. 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, and R. De Nicola. Mobile Distributed Programming in X-Klaim. In *Formal Methods for Mobile Computing*, LNCS 3465, pages 29-68, Springer, 2005.

[4] 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, 1998.

[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. De 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. *Information and Computation*, 202(1): 39–86, 2005.

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

[10] G. Castagna, J. Vitek, and F. Zappa Nardelli. The Seal Calculus. *Information and Computation*, 201(1): 1–54, 2005.

[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 Trans. 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. *Theoretical Computer Science*, 356(3):387–421, 2006.

[15] R. De Nicola, D. Gorla, and R. Pugliese. Basic observables for a calculus for global computing. In *Proc. of ICALP'05*, 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.

[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, 1996.

[19] 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*.

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

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

[22] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322(3): 615–669, 2004.

[23] J. C. Godskesen, and T. Hildebrandt. Extending Howe's Method to Early Bisimulations for Typed Mobile Embedded Resources with Local Names. In *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 140–151. Springer, 2005.

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

[25] M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. *Journal of the ACM*, 52(6): 961–1023, 2005).

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

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

[28] 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.

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

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

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

[32] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis ECS-LFCS-93-266, University of Edinburgh, 1993.

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

[34] A. Schmitt and J.-B. Stefani. The m-calculus: a higher-order distributed process calculus. In *Proc. of POPL'03*, pages 50–61. ACM Press, 2003.

[35] A. Unyapoth and P. Sewell. Nomadic Pict: Correct communication infrastructures for mobile computation. In *Proc. of POPL'01*, pages 116–127. ACM, 2001.