In *Proc. of ICALP'05*, LNCS 3580, pages 1226–1238. © Springer.

Basic Observables for a Calculus for Global Computing

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

[1]Dipartimento di Sistemi e Informatica, Università di Firenze
[2]Dipartimento di Informatica, Università di Roma "La Sapienza"

Abstract. We introduce a foundational language for modelling applications over global computers whose interconnection structure can be explicitly manipulated. Together with process distribution, mobility, remote operations and asynchronous communication through distributed data spaces, the language provides constructs for explicitly modelling inter-node connections and for dynamically establishing and removing them. For the proposed language, we define natural notions of extensional observations and study their closure under operational reductions and/or language contexts to obtain *barbed congruence* and *may testing* equivalence. For such equivalences, we provide alternative characterizations in terms of a labelled *bisimulation* and a *trace* equivalence that can be used for actual proofs.

1 Introduction

In the last decade, we have witnessed the birth of many calculi and kernel languages intended to support programming of global systems and to provide formal tools for reasoning over them. These formalisms in general provide constructs and mechanisms, at different abstraction levels, for modelling the execution contexts of the network where applications roam and run, for coordinating and monitoring the use of resources, for expressing process communication and mobility, and for specifying and enforcing security policies. However, much research effort has been devoted to studying the impact of different communication and mobility paradigms, but little attention has been devoted to the modelling of the actual network underlying global computers as such. Usually, the model of the network implicitly originates from other linguistic choices concerning, e.g., the mobility paradigm. All foundational languages proposed in the literature either model the network as an evolving graph of fully connected nodes [17, 9, 26, 1] or model it as an evolving forest of trees [7, 14, 24, 8]. In our view, both approaches do not convincingly model global computers (the Internet is neither a clique nor a forest of trees) and lack of flexibility ('sharing of resources' is difficult to control and requires complex modelling).

Here, we want to develop the semantic theory of a new model that takes its origin from two formalisms with opposite objectives, namely the programming language X-Klaim [2] and the π-calculus [23]. The former one is a full fledged programming language based on Klaim [9], while the latter one is the generally recognized minimal common denominator of calculi for mobility. The resulting model has been called tKlaim (*topological* Klaim); it retains the main features of Klaim (distribution, remote

Nets: $N \ ::= \ \mathbf{0} \ \big| \ l :: C \ \big| \ \{l_1 \leftrightarrow l_2\} \ \big| \ (\nu l)N \ \big| \ N_1 \| N_2$

Components:
$C \ ::= \ \langle l \rangle \ \big| \ P \ \big| \ C_1|C_2$

Processes:
$P \ ::= \ \mathbf{nil} \ \big| \ a.P \ \big| \ P_1|P_2 \ \big| \ X \ \big| \ \mathbf{rec}\, X.P$

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

Table 1. tKlaim Syntax

operations, process mobility and asynchronous communication through distributed data spaces), but extends it with new constructs to flexibly model the interconnection structure underlying a net. tKlaim permits explicit creation of inter-node connections and their destruction. Connections are essential to perform remote operations: these are possible only if the node where they are initiated and the target one are directly connected.

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

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

Other calculi for global computers make use of (barbed) congruences induced by similar observables: for example, Ambient uses barbs that are somehow related to *i.*; the barbs in Dπ-calculus instead, are strongly related to *iii.*. Within our framework, it can be proved that, by closing observations under any tKlaim context, the three observables all yield the same congruence. This is already an indication of the robustness of the resulting semantic theories. Moreover, the observables are powerful enough to yield interesting theories also when considering lower-level features, such as failures [11].

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

In this paper, we concentrate on the barbed and may equivalences induced by the first basic observable (*a site is up and running*) and establish their correspondence with a bisimulation-based and a trace-based equivalence. To this aim, we introduce a labelled transition system for tKlaim (with labels indicating the performed action) and, on top of it, we define alternative characterizations of barbed congruence and may testing in terms of (non-standard) labelled *bisimilarity* and *trace* equivalence, resp.. The actual development of the alternative characterizations, although performed along the lines of similar results for CCS [20, 4] and π-calculus [23] had to face problems raised by process distribution and mobility, by the explicit use of connections and by asynchrony.

2 The Process Language tKlaim

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

Nets, ranged over by $N, M, \ldots$, 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 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 recursion. *Actions* permit removing/adding data from/to node repositories (actions **in** and **out**), activating new threads of execution (action **eval**), creating new nodes (action **new**), and establishing and removing connections (actions **conn** and **disc**). Notice that $\mathbf{in}(l)@l'$ differs from $\mathbf{in}(!x)@l'$ in that the former evolves only if datum $\langle l \rangle$ is present at l', while the latter accepts any datum. Indeed, $\mathbf{in}(l)@l'$ is a form of *name matching operator* reminiscent of Linda's [16] pattern-matching.

Names occurring in tKlaim processes and nets can be *bound*. More precisely, prefix $\mathbf{in}(!x)@u.P$ binds x in P; prefix $\mathbf{new}(l).P$ binds l in P, and, similarly, net restriction $(\nu l)N$ binds l in N; finally, $\mathbf{rec}\, X.P$ binds X in P. A name that is not bound is called *free*. The sets $fn(\cdot)$ and $bn(\cdot)$ 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* if one can be obtained from the other by renaming bound names. We shall say that a name u is fresh for $\_$ if $u \notin n(\_)$. In the sequel, we shall work with terms whose bound names are all distinct and different from the free ones.

tKlaim operational semantics relies on a structural congruence and a reduction relation. The *structural congruence*, $\equiv$, is formally defined in [10] and identifies nets which intuitively represent the same net. It is inspired to π-calculus's structural congruence (see, e.g., [23]): it states that '$\parallel$' and '$|$' are monoidal operators with **0** and **nil** as identity elements, it equates alpha-equivalent nets, it regulates commutativity of restrictions, and it allows to freely fold/unfold recursive processes. Moreover, the following laws are crucial in our setting:

(Clone) $l :: C_1|C_2 \;\equiv\; l :: C_1 \parallel l :: C_2$

(Self) $l :: \mathbf{nil} \;\equiv\; \{l \leftrightarrow l\}$

(BiDir) $\{l_1 \leftrightarrow l_2\} \;\equiv\; \{l_2 \leftrightarrow l_1\}$

(RNode) $(\nu l)N \equiv (\nu l)(N \parallel l :: \mathbf{nil})$

(Ext) $N \parallel (\nu l)M \equiv (\nu l)(N \parallel M)$ if $l \notin fn(N)$

(Clone) turns the parallel composition of co-located components into a parallel between nodes; (Self) states that nodes are self-connected; (BiDir) states that connections are bidirectional; (Ext) is the standard π-calculus rule for scope extension. Finally, (RNode) states that any restricted name can be used as the address of a node; indeed, we consider restricted names as private network addresses, whose corresponding nodes can be

(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)
$l :: \mathbf{new}(l').P \longmapsto (\nu l')(l :: P \parallel l' :: \mathbf{nil})$

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

(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)
$$\frac{N \equiv M \longmapsto M' \equiv N'}{N \longmapsto N'}$$

Table 2. tKlaim Operational Semantics

activated and deactivated on demand. In the sequel, we shall assume that each restricted name does correspond to an actual node. This assumption is justified by law (RNode).

The reduction relation is given in Table 2. In (R-Out) and (R-Eval), the existence of a connection between the nodes source and target of the action is necessary to place the spawned component. Notice that existence of the connection can only be checked at run-time: an approach like [17] does not fit well in a global computing setting because it relies on a typing mechanism that would require to statically know the whole net. (R-In) and (R-Match) additionally require the existence of a matching datum in the target node. (R-Match) states that action $\mathbf{in}(l)@l_2$ consumes exactly the datum $\langle l \rangle$ at l_2, while (R-In) states that action $\mathbf{in}(!\,x)@l_2$ can consume any $\langle l \rangle$ at l_2; l will then replace the free occurrences of x in the continuation of the process performing the action. (R-New) states that execution of action $\mathbf{new}(l')$ adds a restriction over l' to the net, while creating a node with address l'. Finally, (R-Conn) and (R-Disc) deal with activation/deactivation of connections. In the first case, we need to ensure that the connected nodes do exist; in the second case, we need to check existence of the connection to be removed.

tKlaim adopts a Linda-like [16] communication mechanism: communication is asynchronous and data are anonymous. Indeed, no synchronization takes place between (sending and receiving) processes, because their interactions are mediated by nodes, that act as data repositories. For the sake of simplicity, we only consider monadic data, but the semantic theories we develop could be smoothly extended to deal with tuples of data and with a full-blown Linda-like *pattern matching* mechanism.

3 Observables, Closures and Equivalences

In this section we present both a linear time and a branching time equivalence that yield sensible semantic theories for tKlaim. The approach we follow relies on the definition of an *observation* (also called *barb*) that intuitively formalises the possible interactions of a process. We use observables to define equivalence relations that identify those nets that cannot be taken apart by any basic observation along reductions in any execution context. As usual, $\models\!\Longrightarrow$ denotes the reflexive and transitive closure of $\longmapsto$ and $\widetilde{l}$ denotes a possibly empty set of names.

Definition 1 (Barbs and Contexts). *Predicate $N \downarrow l$ holds true if and only if $N \equiv (\nu\widetilde{l})(N' \parallel l :: \langle l' \rangle)$ for some $\widetilde{l}$, N' and l' such that $l \notin \widetilde{l}$. Predicate $N \Downarrow l$ holds true if and only if $N \models\!\Longrightarrow N'$ for some N' such that $N' \downarrow l$. A* context $C[\cdot]$ *is a* tKlaim *net with an occurrence of a hole* $[\cdot]$ *to be filled in with any net. Formally,*

$$C[\cdot] \quad ::= \quad [\cdot] \quad \big| \quad N \parallel C[\cdot] \quad \big| \quad (\nu l)C[\cdot]$$

We have chosen the basic observables by taking inspiration from those used for the asynchronous π-calculus [23]. One may wonder if our choice is “correct” and argue that there are other alternative notions of basic observables that seem quite natural, as we have discussed in the Introduction. In the full paper [10], we prove that the congruences induced by these alternative observables do coincide. This means that our results are quite independent from the observable chosen and vindicates our choice. Now, we say that 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 $M \models\!\Longrightarrow M'$ and $N' \Re M'$, for some M';

- *context closed*, if $N \Re M$ implies $C[N] \Re C[M]$, for every context $C[\cdot]$.

Our touchstone equivalences should at the very least relate nets with the same observable behaviour; thus, they must be barb preserving. However, an equivalence defined only in terms of this property would be too weak: indeed, the set of barbs of a net may change during computations or when interacting with an external environment. Moreover, for the sake of compositionality, our touchstone equivalences should also be congruences. These requirements lead us to the following definitions.

Definition 2 (May testing). $\simeq$ *is the largest symmetric, barb preserving and context closed relation between nets.*

Definition 3 (Barbed congruence). $\cong$ *is the largest symmetric, barb preserving, reduction and context closed relation between nets.*

We want to remark that the above definition of barbed congruence is the standard one, see [18, 23]. May testing is, instead, usually defined in terms of *observers*, *experiments* and *success of an experiment* [13]. In [10], we prove that, if we let $\simeq'$ denote the equivalence on tKlaim nets defined a lá [13], the two definitions do coincide. Moreover, the inclusions between our touchstone equivalences reflect the inclusions that hold in the π-calculus, since also in our setting may testing, differently from barbed congruence, ignores the branching structure of a process. A pair of nets proving that $\cong\ \subset \simeq$ can be obtained from the CCS terms $a_1.(a_2 + a_3)$ and $a_1.a_2 + a_1.a_3$, that are may testing equivalent but not barbed congruent, by implementing the non-deterministic choice (‘+’) through parallel composition.

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

4 Alternative Characterisations

4.1 A Labelled Transition System

In order to provide more tractable characterisations of our touchstone equivalences, we introduce a labelled transition system (LTS) to make apparent the action a net is willing to perform in order to evolve. The *labelled transition relation*, $\xrightarrow{\alpha}$, is defined as the least relation over nets induced by the inference rules in Table 3. Labels take the form

$$\alpha ::= \tau \;\big|\; l_1 \curvearrowright l_2 \;\big|\; (\nu\widetilde{l})\langle l\rangle @ l_1 : l_2 \;\big|\; l_1 : \triangleright l_2 \;\big|\; l_1 : (\nu\widetilde{l})l \triangleleft l_2 \;\big|\; l_1 : l_2 \;\big|\; l_1 : \neg l_2$$

We let $bn(\alpha)$ be $\widetilde{l}$ if $\alpha = (\nu\widetilde{l})\,\langle l\rangle\,@\,l_1 : l_2$ or $\alpha = l_1 : (\nu\widetilde{l})l \triangleleft l_2$, and be $\emptyset$ otherwise; $fn(\alpha)$ and $n(\alpha)$ are defined accordingly.

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

τ : this means that N may perform a reduction step to become N' (see Proposition 1).

$l_1 \curvearrowright l_2$: this means that in N there is a direct connection between nodes l_1 and l_2 (see (LTS-Link)).

$(\nu\widetilde{l})\,\langle l\rangle\,@\,l_1 : l_2$: this means that in N there is a datum $\langle l\rangle$ located at l_1 and a connection $\{l_1 \leftrightarrow l_2\}$; the datum is available for processes located at l_2 (see (LTS-Datum), (LTS-Offer) and (LTS-Link)). Moreover, according to whether $\widetilde{l} = \{l\}$ or $\widetilde{l} = \emptyset$, we also know if N restricts l or not (see (LTS-Open)).

$l_1 : \triangleright l_2$: this means that in N there is a process located at l_1 willing to send a component at l_2 (see (LTS-Out) and (LTS-Eval)[1]). For the sending to take place, a direct connection between such nodes is needed (see (LTS-Send)).

$l_1 : (\nu\widetilde{l})l \triangleleft l_2$: this means that in N there is a process located at l_1 willing to retrieve a (possibly fresh) datum $\langle l\rangle$ at l_2 (see (LTS-In), (LTS-Match) and (LTS-BIn)). For the actual retrieval, a direct connection between such nodes and a proper datum at l_2 are needed (see (LTS-Comm)).

$l_1 : l_2$: this means that in N there is a process located at l_1 willing to activate a connection with l_2 (see (LTS-Conn)). For the actual activation, the net must contain a node with address l_2, as pointed out by label $l_2 \curvearrowright l_2$ (see (LTS-Est) and (Self)).

$l_1 : \neg l_2$: this means that in N there is a process located at l_1 willing to deactivate a connection with l_2 (see (LTS-Disc)). For the actual deactivation, the net must contain the connection $\{l_1 \leftrightarrow l_2\}$ (see (LTS-Rem)).

The last four kinds of labels describe 'intentions' of a process running in the net. Thus, (LTS-Out) should be read as: "process **out**$(l)@l_2.P$ running at l_1 is willing to send a component at l_2; whenever the execution context provides the connection needed, l_1 will host process P for execution and will run in a net where the connection $\{l_1 \leftrightarrow l_2\}$ does exist and the datum $\langle l\rangle$ is placed at l_2". (LTS-Eval), (LTS-In), (LTS-Match), (LTS-Conn) and (LTS-Disc) should be interpreted similarly.

[1] It should not be surprising that actions **out** and **eval** yield the same label. Of course, the two actions should be taken apart for security reasons because accepting processes for execution is more dangerous than accepting data. However, in our setting, an external observer has not enough power to notice any difference: in both cases, it can just observe that a packet is sent.

(LTS-Out)
$$l_1 :: \mathbf{out}(l)@l_2.P \xrightarrow{l_1 : \triangleright\, l_2} l_1 :: P \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l \rangle$$

(LTS-Eval)
$$l_1 :: \mathbf{eval}(P_2)@l_2.P_1 \xrightarrow{l_1 : \triangleright\, l_2} l_1 :: P_1 \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: P_2$$

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

(LTS-Match)
$$l_1 :: \mathbf{in}(l)@l_2.P \xrightarrow{l_1 : \, l \,\triangleleft\, l_2} l_1 :: P \parallel \{l_1 \leftrightarrow l_2\}$$

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

(LTS-Conn)
$$l_1 :: \mathbf{conn}(l_2).P \xrightarrow{l_1 : l_2} l_1 :: P \parallel \{l_1 \leftrightarrow l_2\}$$

(LTS-Disc)
$$l_1 :: \mathbf{disc}(l_2).P \xrightarrow{l_1 : \neg l_2} l_1 :: P \parallel l_2 :: \mathbf{nil}$$

(LTS-Link) $\{l_1 \leftrightarrow l_2\} \xrightarrow{l_1 \curvearrowright l_2} \mathbf{0}$

(LTS-Datum) $l_1 :: \langle l \rangle \xrightarrow{\langle l \rangle\ @\ l_1 : l_1} \mathbf{0}$

(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-BIn)
$$\frac{N \xrightarrow{l_1 : \, l \,\triangleleft\, l_2} N' \qquad l \notin fn(N)}{N \xrightarrow{l_1 : (\nu l)\ l \,\triangleleft\, l_2} N' \parallel l :: \mathbf{nil}}$$

(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-Send)
$$\frac{N_1 \xrightarrow{l_1 : \triangleright\, l_2} N_1' \qquad N_2 \xrightarrow{l_1 \curvearrowright l_2} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

(LTS-Comm)
$$\frac{N_1 \xrightarrow{l_1 : \, l \,\triangleleft\, l_2} N_1' \qquad N_2 \xrightarrow{\langle l \rangle\ @\ l_2 : l_1} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

(LTS-Est)
$$\frac{N_1 \xrightarrow{l_1 : l_2} N_1' \qquad N_2 \xrightarrow{l_2 \curvearrowright l_2} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

(LTS-Rem)
$$\frac{N_1 \xrightarrow{l_1 : \neg l_2} N_1' \qquad N_2 \xrightarrow{l_1 \curvearrowright l_2} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

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

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

Table 3. A Labelled Transition System

(LTS-Open) signals extrusion of bound names; as in some presentation of the π-calculus, this rule is used to investigate the capability of processes to export bound names, rather than to actually extend the scope of bound names.

Notice that the LTS of Table 3 may appear unnecessarily complicated as a tool to define the operational semantics of tKlaim: consider, e.g., the right hand side of the rules for **out**/**in**/**eval**, or rule (LTS-BIn) (used to signal that a received name is fresh for the receiving net). Nevertheless, it is adequate as a tool to establish alternative, more tractable, characterisations of the touchstone equivalences we are interested in. Indeed, the complications in the operational rules of Table 3 resemble those arisen in [25] when defining an ‘equivalent’ LTS depending on the reduction semantics of a calculus. However, in [25] only simple calculi are considered and it would be interesting to investigate if the approach can be satisfactory extended to tKlaim. Finally, the LTS is ‘correct’ w.r.t. the actual operational semantics of tKlaim, $\longmapsto$, as stated by the following Proposition.

Proposition 1. *$N \longmapsto M$ if and only if $N \xrightarrow{\tau} M$.*

4.2 Bisimulation Equivalence

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

$$\chi \ ::= \ \tau \ \Big| \ l_1 \curvearrowright l_2 \ \Big| \ (\nu\widetilde{l})\,\langle l\rangle\ @\ l_1 \ : \ l_2$$

Moreover, $\Rightarrow$ stands for $\xrightarrow{\tau}{}^{*}$, $\overset{\alpha}{\Longrightarrow}$ stands for $\Rightarrow \xrightarrow{\alpha} \Rightarrow$, and $\overset{\hat{\alpha}}{\Longrightarrow}$ stands for $\Rightarrow$, if $\alpha = \tau$, and for $\overset{\alpha}{\Longrightarrow}$, otherwise.

Definition 4 (Bisimilarity). *A symmetric relation $\mathfrak{R}$ between* tKlaim *nets is a* (weak) bisimulation *if, for each $N\ \mathfrak{R}\ M$, it holds that:*

1. *$N \xrightarrow{\chi} N'$ implies that $M \overset{\hat{\chi}}{\Longrightarrow} M'$ and $N'\ \mathfrak{R}\ M'$, for some M';*
2. *$N \xrightarrow{l_1 :\, \triangleright\, l_2} N'$ implies that $M \parallel \{l_1 \leftrightarrow l_2\} \Rightarrow M'$ and $N'\ \mathfrak{R}\ M'$, for some M';*
3. *$N \xrightarrow{l_1 :\, l\, \triangleleft\, l_2} N'$ implies that $M \parallel \{l_1 \leftrightarrow l_2\} \parallel l_2 :: \langle l\rangle \Rightarrow M'$ and $N'\ \mathfrak{R}\ M'$, for some M';*
4. *$N \xrightarrow{l_1 \,:\, l_2} N'$ implies that $M \parallel l_2 :: \mathbf{nil} \Rightarrow M'$ and $N'\ \mathfrak{R}\ M'$, for some M';*
5. *$N \xrightarrow{l_1 \,:\, \neg l_2} N'$ implies that $M \parallel \{l_1 \leftrightarrow l_2\} \Rightarrow M'$ and $N'\ \mathfrak{R}\ M'$, for some M'.*

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

Bisimilarity requires that labels of the form $(\nu\widetilde{l})\,\langle l\rangle\ @\ l_1 \ : \ l_2$ or $l_1 \curvearrowright l_2$ must be replied to with the same label (possibly with some additional τ-step). This is necessary since such labels describe the structure of the net (its data and connections) and, to be equivalent, two nets must have at least the same structure. Labels different from χ only express intentions and are handled differently. For example, the intention of sending a component, say $N \xrightarrow{l_1 :\, \triangleright\, l_2} N'$, can be simulated by a net M (in a context where l_1 and l_2 are connected) through the execution of some τ-steps that lead to some M' equivalent to N'. Indeed, since we want our bisimulation to be a congruence, a context that provides a connection between the source and the target nodes of the sending action must not tell apart N and M. Similar considerations also hold for the last three items of Definition 4.

Notice that labels of the form $l_1 : (\nu l) l \triangleleft l_2$ are not necessary for the definition of bisimulation. Indeed, they exactly work like labels $l_1 : l \triangleleft l_2$ with the extra information that l is fresh for the receiving net; for the bisimulation game this information is useless, while it will be of fundamental importance when considering trace-based equivalence.

Remarkably, though in tKlaim processes can occur as arguments in process actions (**eval**), the LTS and the bisimulation we developed do not use labels containing processes. Thus, the bisimulation relies only on a standard quantification over names (in the input case) and we strongly conjecture that it is decidable, under proper assumptions: techniques similar to those in [21] could be used here. Moreover, the presence of rule (LTS-Struct) in the LTS does not compromise the tractability of $\approx$; obviously, (LTS-Struct) can be dropped, if one is prepared to have more rules in the LTS.

We can now present our first main result, whose proof is in the full paper [10].

Theorem 1 (Alternative Characterization of Barbed Congruence). $\approx \; = \; \cong$.

4.3 Trace Equivalence

In this section, we develop an alternative characterization of may testing. For some well-known process calculi, may testing coincides with trace equivalence [13, 3, 5]; in this section, we show how a similar result is obtained also in the setting of tKlaim.

The idea behind trace equivalence is that N and M are related if and only if the sets of their traces coincide. Put in another form, if N exhibits a sequence of visible actions σ, then M must exhibit σ as well, and vice versa. In an asynchronous setting [5], this requirement must be properly weakened, since the discriminating power of asynchronous contexts is weaker: in the asynchronous π-calculus, for example, contexts cannot observe input actions.

To carry out proofs, we found it convenient to introduce a *complementation function* $\overline{\cdot}$ over visible labels (i.e. labels different from τ), ranged over by ϕ, such that

$$\overline{l_1 \curvearrowright l_2} \triangleq l_1 : \triangleright l_2 \qquad \overline{l_1 : \triangleright l_2} \triangleq l_1 \curvearrowright l_2 \qquad \overline{l_1 : (\nu\widetilde{l})\, l \triangleleft l_2} \triangleq (\nu\widetilde{l})\, \langle l \rangle @ l_2 : l_1$$

$$\overline{l_1 : l_2} \triangleq l_2 \curvearrowright l_2 \qquad \overline{l_1 : \neg l_2} \triangleq l_1 \curvearrowright l_2 \qquad \overline{(\nu\widetilde{l})\, \langle l \rangle @ l_2 : l_1} \triangleq l_1 : (\nu\widetilde{l})\, l \triangleleft l_2$$

Because of the interplay between free and bound names (bound names are always associated to nodes, see rule (RNode)), we need to distinguish reception of a free name from reception of a bound name (that must be fresh for the receiving net). Similarly to the π-calculus [3, 5], this can be done by exploiting *bound input* labels, $l_1 : (\nu l) l \triangleleft l_2$, generated by rule (LTS-BIn) (that also adds a node with address l because of law (RNode)). Finally, we let σ to range over (possibly empty) sequences of visible actions, i.e.

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

where ϵ denotes the empty sequence of actions and '$\cdot$' represents concatenation. As usual, $N \overset{\epsilon}{\Longrightarrow}$ denotes $N \Rightarrow$ and $N \overset{\phi\cdot\sigma}{\Longrightarrow}$ denotes $N \overset{\phi}{\Rightarrow} \overset{\sigma}{\Longrightarrow}$.

The naive formulation of trace equivalence such as "$N \overset{\sigma}{\Longrightarrow}$ if and only if $M \overset{\sigma}{\Longrightarrow}$" is too strong in an asynchronous setting: for example, it would distinguish $l :: \mathbf{in}(!x)@l_1.\mathbf{in}(!y)@l_2$ and $l :: \mathbf{in}(!y)@l_2.\mathbf{in}(!x)@l_1$, which are indeed may testing equivalent. Like in [5], a weaker trace-based equivalence can be defined as follows.

Definition 5 (Trace Equivalence). *$\asymp$ is the largest symmetric relation between* tKlaim *nets such that, whenever $N \asymp M$, it holds that $N \overset{\sigma}{\Longrightarrow}$ implies $M \overset{\sigma'}{\Longrightarrow}$, for some $\sigma' \preceq \sigma$.*

The crux is to identify a proper ordering on the traces such that may testing is exactly captured by $\asymp$. The ordering $\preceq$ is the least reflexive and transitive relation induced by the laws in Table 4. The first three laws have been inspired by [5], while the last five

(L1)	$\sigma \cdot (\nu\widetilde{l})\sigma' \leq \sigma \cdot (\nu\widetilde{l})(\beta \cdot \sigma')$	if $(\nu\widetilde{l})\sigma' \neq \mathit{UNDEF}$
(L2)	$\sigma \cdot (\nu\widetilde{l})(\phi \cdot \gamma \cdot \sigma') \leq \sigma \cdot (\nu\widetilde{l})(\gamma \cdot \phi \cdot \sigma')$	if $(\nu\widetilde{l})(\phi \cdot \gamma \cdot \sigma') \neq \mathit{UNDEF}$
(L3)	$\sigma \cdot (\nu\widetilde{l})\sigma' \leq \sigma \cdot (\nu\widetilde{l})(\gamma \cdot \overline{\gamma} \cdot \sigma')$	if $(\nu\widetilde{l})\sigma' \neq \mathit{UNDEF}$
(L4)	$\sigma \cdot l : \triangleright l \cdot \phi \cdot \sigma' \leq \sigma \cdot \phi \cdot \sigma'$	if $l \in \Upsilon(\phi)$
(L5)	$\sigma \cdot l : \triangleright l \cdot \phi \cdot \sigma' \leq \sigma \cdot \phi \cdot l : \triangleright l \cdot \sigma'$	if $l \notin bn(\phi)$
(L6)	$\sigma \cdot \phi' \cdot \sigma' \leq \sigma \cdot \phi \cdot \sigma'$	if $(\phi, \phi') \in \Psi$
(L7)	$\sigma \cdot l_1 : (\nu\widetilde{l})\, l \triangleleft l_1 \cdot \sigma' \leq \sigma \cdot l_2 : (\nu\widetilde{l})\, l \triangleleft l_1 \cdot \sigma'$	
(L8)	$\sigma \cdot l_2 : (\nu\widetilde{l})\, l \triangleleft l_1 \cdot \sigma' \leq \sigma \cdot l_1 : (\nu\widetilde{l})\, l \triangleleft l_1 \cdot l_1 : \triangleright l_2 \cdot \sigma'$	

In law (L1), β stands for either $l_1 : \triangleright l_2$ or $l_1 : l \triangleleft l_2$ or $l_1 : l_2$ or $l_1 : \neg l_2$.
In laws (L2) and (L3), γ stands for either $l_1 : \triangleright l_2$ or $l_1 : l \triangleleft l_2$.
In law (L4), function $\Upsilon(\cdot)$ is defined as follows: $\Upsilon(l_1 : (\nu\widetilde{l})l \triangleleft l_2) = \Upsilon(l_1 : l_2) = \{l_1, l_2\}$ and $\Upsilon(l_1 \curvearrowright l_2) = \{l_1\}$ and $\Upsilon(l_1 : \triangleright l_2) = \Upsilon((\nu\widetilde{l})\,\langle l \rangle\, @\, l_1 : l_2) = \{l_2\}$.
In law (L6), relation Ψ is $\{ (l_1 : \neg l_2, l_1 : \triangleright l_2), (l_1 : \triangleright l_2, l_1 : \neg l_2), (l_2 : \triangleright l_2, l_1 : l_2) \}$.

Table 4. The Ordering Relation on Traces

ones are strictly related to inter-node connections. 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. The ordering $\leq$ relies on the function $(\nu\widetilde{l})\sigma$, that is used in laws (L1), (L2) and (L3) when moving/removing a label of the form $l_1 : (\nu l)\, l \triangleleft l_2$. In this case, the information that l is a fresh received value must be kept in the remaining trace. The formal definition is

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

$$(\nu l)(\phi \cdot \sigma) \triangleq \begin{cases} l_1 : (\nu l)\, l \triangleleft l_2 \cdot \sigma & \text{if } \phi = l_1 : l \triangleleft l_2 \text{ and } l \notin \{l_1, l_2\} \\ \phi \cdot (\nu l)\sigma & \text{if } l \notin n(\phi) \text{ and } (\nu l)\sigma \neq \mathit{UNDEF} \\ \mathit{UNDEF} & \text{otherwise} \end{cases}$$

Further explanations can be found in the full paper [10].

The rules in Table 4 can be explained as follows. (L1) states that labels representing intentions cannot be directly observed; at most, their effect can be somehow observed. (L2) states that the execution of an input/output/migration can be delayed along computations without being noticed by any observer. (L3) states that two adjacent 'complementary' actions can be deleted. (L4) states that an action involving l as source or target node always enables sending actions from l to l; because, in all these cases, a node at address l exists. Function $\Upsilon(\cdot)$ is needed to restrict applicability of (L4) only to the cases needed to prove Theorem 2. (L5) states that, if a sending action from l to l is enabled after an action ϕ, then the action can take place before ϕ, since the node at l was already present; clearly, this is possible only if l is not bound by ϕ. (L6) states that some intentions are interchangeable; indeed, since the complementation function is not injective, the same observer may enable different kinds of process actions. (L7) states that, if a process located at l_2 can retrieve a datum from l_1, then processes located at l_1 can retrieve such datum as well. Finally, (L8) states that, if a process located at l_1

can retrieve a datum $\langle l \rangle$ locally and then migrate at l_2, then processes located at l_2 can retrieve $\langle l \rangle$ remotely.

Remarkably, may testing in the (synchronous/asynchronous) π-calculus [3, 5] cannot distinguish bound names from free ones; thus, a bound name can be replaced with any name in a trace. This is *not* the case here: indeed, bound names can always be considered as addresses of nodes, while free names cannot. This makes a difference for an external observer; thus, a law like

$$\sigma \cdot \langle l' \rangle @ l_1 : l_2 \cdot (\sigma'[^{l'}/_{l}]) \leq \sigma \cdot (\nu l) \langle l \rangle @ l_1 : l_2 \cdot \sigma'$$

(that, mutatis mutandis, holds for the π-calculus [3, 5]) does not hold for TKLAIM.

We can now state our second major result; detailed proofs are in [10].

Theorem 2 (Alternative Characterization of May Testing). $\asymp = \simeq$.

5 Conclusions and Related Work

We have introduced TKLAIM, a foundational language that provides constructs to explicitly model and dynamically establish/remove inter-node connections, and some associated semantic theories. In a companion paper [11], we have applied the theory to a few examples that illustrate usability.

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. First, TKLAIM clearly enlightens the key features we want to model such as distribution and mobility of processes, and inter-node connections; an encoding of such features in the π-calculus would hide them within complex process structures. Second, a convincing encoding should enjoy 'reasonable' properties, like those pointed out in [22]. We believe this is *not* the case. For example, in [12] we developed an intuitive encoding of a TKLAIM's sub-calculus into the asynchronous π-calculus that does not preserve convergence. We are now working on proving that this is not incidental and is due to the check of existence of the target of a communication that is performed in TKLAIM and not in the π-calculus. We conjecture that a divergence free encoding does not exist.

Related Work. To our knowledge, no alternative characterization of may testing in terms of a trace-based equivalence has ever been given for a distributed language with process mobility. Bisimulation-based equivalences for calculi relying on a flat net topology are developed in [1, 17]; such equivalences are mainly derived from bisimulation equivalences for the π-calculus and its variants. Bisimulation-based equivalences for calculi relying on a hierarchical net topology are developed in [19, 6, 8]. Although these bisimulations are inspired by Sangiorgi's *context bisimulation* [23] and, thus, exploit universal quantification over processes, they yield proof techniques that are usable in practice.

Finally, the most closely related work is [15]; 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. However, differently from our approach, the authors only consider links that can fail and do not model dynamic connections establishment.

Acknowledgements. We thank the anonymous referees for their useful comments.

References

1. R. M. Amadio. On modelling mobility. *Theor. Comp. Sci.*, 240(1):147–176, 2000.
2. 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.
3. M. Boreale and R. De Nicola. Testing equivalences for mobile processes. *Information and Computation*, 120:279–303, 1995.
4. M. Boreale, R. De Nicola, and R. Pugliese. Basic observables for processes. *Information and Computation*, 149(1):77–98, 1999.
5. M. Boreale, R. De Nicola, and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 172:139–164, 2002.
6. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication and Mobility Control in Boxed Ambients. To appear in *Information and Computation*.
7. L. Cardelli and A. D. Gordon. Mobile ambients. *TCS*, 240(1):177–213, 2000.
8. G. Castagna and F. Zappa Nardelli. The Seal Calculus Revisited: contextual equivalence and bisimilarity. In *Proc. of FSTTCS'02*, volume 2556 of *LNCS*, pages 85–96.
9. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.
10. R. De Nicola, D. Gorla, and R. Pugliese. Basic observables for a calculus for global computing. Tech. Rep. 07/2004, Dip. Informatica, Università di Roma "La Sapienza", 2004.
11. R. De Nicola, D. Gorla, and R. Pugliese. Global Computing in a Dynamic Network of Tuple Spaces. In *Proc. of COORDINATION'05*, volume 3454 of *LNCS*, pages 157–172.
12. R. De Nicola, D. Gorla, and R. Pugliese. On the Expressive Power of KLAIM-based Calculi. *Proc. of EXPRESS'04*, ENTCS 128(2):117–130.
13. R. De Nicola and M. Hennessy. Testing equivalence for processes. *TCS*, 34:83–133, 1984.
14. C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *Proc. of CONCUR '96*, volume 1119 of *LNCS*, pages 406–421. Springer, 1996.
15. A. Francalanza and M. Hennessy. A Theory of System Behaviour in the Presence of Node and Link Failures. Tech. Rep. cs01:2005, Univ. of Sussex.
16. D. Gelernter. Generative communication in linda. *TOPLAS*, 7(1):80–112. ACM, 1985.
17. M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. In *FoSSaCS '03*, volume 2620 of *LNCS*, pages 282–299.
18. K. Honda and N. Yoshida. On reduction-based process semantics. *TCS*, 152(2), 1995.
19. M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. In *Proc. of ICALP'03*, volume 2719 of *LNCS*, pages 584–598. Springer, 2003.
20. R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.
21. 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.
22. C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.
23. D. Sangiorgi and D. Walker. The π-calculus: a Theory of Mobile Processes. *Cambridge University Press*, 2001.
24. A. Schmitt and J.-B. Stefani. The m-calculus: a higher-order distributed process calculus. *SIGPLAN Not.*, 38(1):50–61, 2003.
25. P. Sewell. From Rewrite Rules to Bisimulation Congruences. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*, pages 269–284. Springer, 1998.
26. P. Sewell, P. Wojciechowski, and B. Pierce. Location independence for mobile agents. In *Proc. of ICCL*, volume 1686 of *LNCS*. Springer, 1999.