Theoretical Computer Science, 356(3):387–421. © Elsevier, 2006.

On the Expressive Power of KLAIM-based Calculi ⋆

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 study the expressive power of variants of KLAIM, an experimental language with programming primitives for network-aware programming that combines the process algebra approach with the coordination-oriented one. KLAIM has proved to be suitable for programming a wide range of distributed applications with agents and code mobility, and has been implemented on the top of a runtime system written in Java. In this paper, the expressivity of its constructs is tested by distilling from it a few, more and more foundational, languages and by studying the encoding of each of them into a simpler one. The expressive power of the considered calculi is finally tested by comparing one of them with asynchronous π-calculus.

Key words: Process calculi, Network-aware programming, Expressiveness, Language encodings, Behavioural equivalences, Bisimulation

1 Introduction

In the design of programming languages for network-aware programming, a key research challenge is devising theoretical models and calculi with a clean formal semantics for specifying, programming and reasoning about network-aware applications. These models and calculi could provide the basis for the design of systems sound "by construction" and behaving in a predictable and analyzable way. The crux is to identify the more appropriate abstractions and to supply foundational and effective tools for supporting the development of network-aware applications.

⋆ This work is the full version of [14] and has been carried on while the second author was a PhD student at the University of Florence.

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

One of the abstractions that appears to be very important is *mobility*. This feature deeply increases flexibility and, thus, expressiveness of programming languages for network-aware programming. Evidence of the success of this programming style is provided by the recent design of commercial/prototype programming languages with primitives for moving code and processes.

The first foundational calculus dealing with mobility has been the π-calculus [23], a simple and expressive calculus aiming at capturing the essence of name passing with the minimum number of basic constructs. Indeed, the only operators of the π-calculus are the empty process, output and input prefix, parallel composition, name restriction and process replication; the exchanged values of the calculus are just names. If considered from a network-aware perspective, one could say that π-calculus misses an explicit notion of locality and/or domain where computations take place.

To deal with this deficiency of π-calculus, several foundational formalisms, presented as process calculi or strongly based on them, have been developed. They have, undoubtedly, improved the formal understanding of network-aware systems. We want to mention, among the others, Ambient calculus [10], Dπ-calculus [20] and KLAIM [11]. As usual, a major problem in the development of a foundational language is to find appropriate sets of abstractions that can be considered an acceptable compromise between expressiveness, elegance and implementability. A paradigmatic example is the Ambient calculus: it is very elegant and expressive, but it still lacks a reasonable distributed implementation.

We have been long working with KLAIM, an experimental language with programming constructs for network-aware programming that combines the process algebra paradigm with the coordination-oriented one. KLAIM has been specifically designed to program distributed systems consisting of several mobile components that interact through multiple distributed tuple spaces. KLAIM primitives allow programmers to distribute and retrieve data and processes to and from the nodes of a net. Moreover, localities are first-class citizens that can be dynamically created and communicated over the network. Components, both stationary and mobile, can explicitly refer and control the spatial structures of the network. Communication takes place through distributed repositories (a very flexible model that meets important requirements of network-aware programming) and remote operations (to supply a realistic abstraction level and avoid heavily resorting to code mobility).

KLAIM rests on an extension of the basic LINDA coordination model [17] with multiple distributed tuple spaces. A *tuple space* is a multiset of *tuples* that are sequences of information items. Tuples are anonymous and can be associatively selected from tuple spaces by means of a *pattern-matching* mechanism. Tuples can contain both values and code that can be subsequently accessed and evaluated. An allocation environment (associating logical and physical localities) is used to avoid the programmers to consider the precise physical allocation of the distributed tuple spaces.

KLAIM has been upgraded to a full fledged programming language (called X-KLAIM [2,4]) by relying on the implementation of a run-time system [3] developed in Java for the sake of portability. The linguistic constructs of KLAIM have proved to be appropriate for programming a wide range of distributed applications with agents and code mobility [11,12] that, once compiled in Java, can be run over different platforms.

In this paper, we aim at assessing the expressive power of tuple based communications and evaluating the theoretical impact of the linguistic primitives proposed for KLAIM. This task is performed by distilling from KLAIM a few, more and more, foundational calculi and studying the possibility of encoding each of the calculi in a more basilar one. A tight comparison between these calculi and asynchronous π-calculus [21,6] is also provided. The first sub-calculus we consider is μKLAIM [19]; it is obtained by eliminating from KLAIM the distinction between logical and physical localities (i.e., *no allocation environment*) and the possibility of higher order communication (*i.e., no process code in tuples*). The second sub-calculus, cKLAIM, is obtained from μKLAIM by only considering *monadic* communications and by removing the basic actions **read**. The last calculus, lcKLAIM, is obtained by removing also the possibility of performing remote inputs and outputs; communications is only local and process migration is needed to use remote resources.

To assess the quality of our encodings, we shall use well-established criteria, namely *full abstraction* and *semantical equivalence*, based on an appropriate family of equivalences EQ (see, e.g., [25]).

Full Abstraction w.r.t. EQ**:** An encoding $enc(\cdot)$ of language $\mathcal{X}$ into language $\mathcal{Y}$ satisfies this property if for every pair of $\mathcal{X}$-terms T_1 and T_2 it holds that $T_1\ EQ_{\mathcal{X}}\ T_2$ if and only if $enc(T_1)\ EQ_{\mathcal{Y}}\ enc(T_2)$.

Semantical Equivalence w.r.t. EQ**:** An encoding $enc(\cdot)$ of language $\mathcal{X}$ into language $\mathcal{Y}$ satisfies this property if for every $\mathcal{X}$-term T it holds that $T\ EQ_{\mathcal{Z}}\ enc(T)$, for some language $\mathcal{Z}$ containing both $\mathcal{X}$ and $\mathcal{Y}$.

In the above definitions, EQ is not a precise equivalence but a *family* of equivalences that has to be properly instantiated to the considered languages, say $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$, to obtain $EQ_{\mathcal{X}}$, $EQ_{\mathcal{Y}}$ and $EQ_{\mathcal{Z}}$. Of course, a stronger equivalence guarantees a better encoding, in that it attests that the target language has expressive power closer to that of the source calculus. Moreover, we have that, if an encoding is semantical equivalent w.r.t. EQ then it is also fully abstract w.r.t. the same equivalence. Thus, an encoding enjoying semantical equivalence is 'better' then an encoding enjoying fully abstraction.

The equivalences we use in this paper are barbed bisimilarity, $\dot{\cong}$, and barbed congruence, $\cong$; these are uniformly defined equivalences on process calculi often used as 'touchstone' semantic theories. Barbed bisimulation equates two terms that offer the same observable behaviour along all possible computations. Barbed congruence

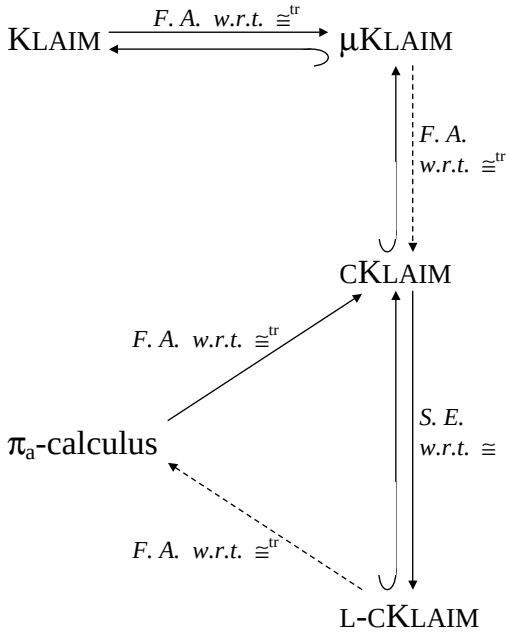

Table 1
Overview of our Results

is obtained by closing barbed bisimulation under all possible language contexts. As expected, see e.g. [30], barbed bisimilarity is coarser then barbed congruence. It often turns out that a 'half-way' solution between the two notions above is the appropriate one; it relies on what we call *translated barbed congruence*, written $\cong^{\text{tr}}$. We say that an encoding $enc(\cdot)$ from language $\mathcal{X}$ to language $\mathcal{Y}$ is fully abstract w.r.t. $\cong^{\text{tr}}$ whenever the set of contexts in $\mathcal{Y}$ considered for context closure is formed by using only the translation via $enc(\cdot)$ of contexts in $\mathcal{X}$. Indeed, if we consider the encoding as a protocol (i.e. a precise sequence of message exchanges), translated contexts represent opponents conforming to the protocol. This result suffices to assess expressiveness of languages, see, e.g., [5,9]. Indeed, it amounts to saying that the source language can be faithfully compiled in the target one.

The main results of our work are summarized in Table 1. There, a labelled arrow between two calculi, $\mathcal{X} \xrightarrow{\mathcal{P}} \mathcal{Y}$, means that language $\mathcal{X}$ can be encoded in language $\mathcal{Y}$ and that the encoding enjoys property $\mathcal{P}$. The arrow is dotted $\dashrightarrow$ if the actual encoding can introduce divergence, i.e. infinite sequences of reductions that in the source term were not present. Moreover, $\hookrightarrow$ stands for the identity encoding, *F.A.* stands for Fully Abstract, and *S.E.* stands for Semantically Equivalent.

The rest of this paper is organized as follows. KLAIM and the three calculi derived from it are presented in Section 2. Sections 3, 4 and 5 present the encodings of

Nets:	N	$::= \mathbf{0} \mid l ::_\rho C \mid N_1 \parallel N_2 \mid (\nu l)N$
Components:	C	$::= \langle t \rangle \mid P \mid C_1 \mid C_2$
Processes:	P	$::= \mathbf{nil} \mid a.P \mid P_1 \mid P_2 \mid X \mid \mathbf{rec}\, X.P$
Actions:	a	$::= \mathbf{in}(T)@u \mid \mathbf{read}(T)@u \mid \mathbf{out}(t)@u \mid \mathbf{eval}(P)@u \mid \mathbf{new}(l)$
Tuples:	t	$::= u \mid P \mid t_1, t_2$
Templates:	T	$::= u \mid !\, x \mid !\, X \mid T_1, T_2$

Table 2
KLAIM syntax

KLAIM in μKLAIM, of μKLAIM into cKLAIM and of cKLAIM into lcKLAIM, respectively. Section 6 contains a comparison with π_a-calculus; in particular it presents an encoding of π_a-calculus into cKLAIM and an encoding of lcKLAIM into π_a-calculus. Section 7 contains a conclusive assessment of the presented encodings, while Section 8 ends the paper.

2 A Family of Process Languages

In this section, we formally present the languages we shall work with, namely KLAIM [11] and the three calculi derived from it.

2.1 KLAIM*: Kernel Language for Agents Interaction and Mobility*

The syntax of KLAIM is given in Table 2. We assume two disjoint countable sets: $\mathcal{L}$ of *names* $l, l', \ldots$ and $\mathcal{V}$ of *variables* $x, y, \ldots, X, Y, \ldots,$ `self`, where `self` is a reserved variable (see below). Notationally, we prefer letters $x, y, \ldots$ when we want to stress the use of a name as a basic variable, and $X, Y, \ldots$ when we want to stress the use of a name as a process variable. We will use u for basic variables and localities.

Processes, ranged over by $P, Q, R, \ldots$, are the KLAIM active computational units and may be executed concurrently either at the same locality or at different localities. Processes are built from the terminated process **nil** and from basic actions by using action prefixing, parallel composition and recursion. *Basic Actions*, ranged over by a, permit removing/accessing/adding data from/to node repositories, activating new threads of execution and creating new nodes. Action **new** is not indexed with an address because it always acts locally; all the other actions explicitly indicate

the (possibly remote) locality where they will take effect. *Tuples*, t, are the communicable objects: they are sequences of names and processes. *Templates*, T, are patterns used to retrieve tuples and the pattern matching underlying the communication mechanism is that of LINDA [17].

Nets, ranged over by $N, M, H, K, \ldots$, are finite collections of nodes. A *node* is a triple $l ::_\rho C$, where locality l is the address of the node, ρ is the *allocation environment* (a finite partial function mapping variables into names, used to implement dynamic binding of names) and C is the component located at l. *Components*, ranged over by $C, D, \ldots$, can be either processes or data, denoted by $\langle t \rangle$. In the net $(\nu l)N$, the scope of the name l is restricted to N; the intended effect is that if one considers the net $N_1 \parallel (\nu l)N_2$ then locality l of N_2 cannot be immediately referred to from within N_1. We say that a net is *well-formed* if for each node $l ::_\rho C$ we have that $\rho(\texttt{self}) = l$, and, for any pair of nodes $l ::_\rho C$ and $l' ::_{\rho'} C'$, we have that $l = l'$ implies $\rho = \rho'$. Hereafter, we will only consider well-formed nets. Moreover, we shall always assume that bound names are always the address of a node in the net.

Names and variables occurring in KLAIM processes and nets can be *bound*. More precisely, prefix **new**$(l).P$ binds name l in P, and, similarly, net restriction $(\nu l)N$ binds l in N. Prefix **in**$(\ldots, !\_, \ldots)@u.P$ binds variable $\_$ in P; this prefix is similar to the λ-abstraction of the λ-calculus. Finally, **rec** $X.P$ binds variable X in P. A name/variable that is not bound is called *free*. The sets $fn(\cdot)$ and $bn(\cdot)$ (respectively, of free and bound names of a term) and $fv(\cdot)$ and $bv(\cdot)$ (of free/bound variables) are defined accordingly. The set $n(\cdot)$ is the union of the free and bound names and variables occurring in $\cdot$. Moreover, we define $fl(N)$ as the subset of $fn(N)$ that are addresses of nodes in N.

As usual, we say that two terms are *alpha-equivalent*, written $=_\alpha$, if one can be obtained from the other by renaming bound names/variables. We shall say that u is fresh for $\_$ if $u \notin n(\_)$. In the sequel, we shall work with terms whose bound variables are all distinct and whose bound names are all distinct and different from the free ones.

Remark 2.1 The language presented so far slightly differs from [11]. The three differences are: the absence of values and expressions, the absence of non-deterministic choice, and the use of recursion instead of process definitions. Values and expressions (e.g., integers, strings, ...) are not included only to simplify reasoning: they can be easily encoded by following the classical implementations in π-calculus (see, e.g., [30]). A restricted form of non-deterministic choice is implicitly provided by KLAIM through actions **read**/**in**: their semantics is determined by the availability of tuples matching a given template, and in case of multiple matching the choice is internally determined. Other forms of choice could be implemented by following [25,24]. Recursion is easier to deal with in a theoretical framework because the syntax of a recursive term already contains all the code needed to properly run the term itself.

Notation 2.2 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 sequences of objects (e.g. $\widetilde{l}$ is a sequence of names); this will be sometimes written as $\widetilde{x}_{i\in I}$, for an appropriate index-set I. Moreover, if $\widetilde{x} = (x_1, ..., x_n)$, we shall assume that $x_i \neq x_j$ for $i \neq j$. If $\widetilde{x} = (x_1, \ldots, x_n)$ and $\widetilde{y} = (y_1, \ldots, y_m)$ then $\widetilde{x}, \widetilde{y}$ will denote the sequence of pairwise distinct elements $(x_1, \ldots, x_n, y_1, \ldots, y_m)$. When convenient, we shall regard a sequence simply as a set.
We shall sometimes write **in**()@l, **out**()@l and $\langle\rangle$ to mean that the argument of the actions or the datum are an empty sequence of items. We usually omit trailing occurrences of process **nil** and write $\Pi_{j\in J}\, W_j$ for the parallel composition (both '|' and '||') of terms (components or nets, resp.) W_j.
Finally, we assume that allocation environments act as the identity on locality names. This assumption simplifies the operational semantics.

The operational semantics relies on a *structural congruence* relation, $\equiv$, bringing the participants of a potential interaction to contiguous positions, and a *reduction relation*, $\longmapsto$, expressing the evolution of a net. The structural congruence is the least congruence closed under the axioms given in the upper part of Table 3. Most of the laws are mundane [22,30], while laws (Abs) and (Clone) are peculiar to our setting. The first one states that **nil** is the identity for '|'; the second one turns a parallel between co-located components into a parallel between nodes (thus, it is also used to achieve commutativity and associativity of '|').

The reduction relation is given in the lower part of Table 3. There, we use two auxiliary functions:

(1) a *tuple/template evaluation* function, $\mathcal{E}[\![\ \_\ ]\!]_\rho$, to transform variables according to the allocation environment of the node performing the action whose argument is $\_$. The main clauses of its definition are given below:

$$\mathcal{E}[\![\ u\ ]\!]_\rho = \begin{cases} u & \text{if } u \in \mathcal{L} \\ \rho(u) & \text{if } u \in dom(\rho) \\ \text{UNDEF} & \text{otherwise} \end{cases} \qquad \mathcal{E}[\![\ P\ ]\!]_\rho = P\{\rho\}$$

where $P\{\rho\}$ denotes the process obtained from P by replacing any free occurrence of a variable x that is not within the argument of an **eval** with $\rho(x)$. Clearly, $\mathcal{E}[\![\ P\ ]\!]_\rho$ is UNDEF if $\rho(x)$ is undefined for some of these x. We shall write $\mathcal{E}[\![\ t\ ]\!]_\rho = t'$ to denote that the evaluation of t using ρ succeeds and returns t'.

(2) a *pattern matching* function, $match(\cdot, \cdot)$, to verify the compliance of a tuple w.r.t. a template and to associate values (i.e. names and processes) to variables bound in templates. Intuitively, a tuple matches against a template if they have the same number of fields, and corresponding fields match (where a bound name matches any value, while two names match only if they are identical).

Axioms for Structural Congruence:

Monoid laws for "||", i.e.

$$N \parallel \mathbf{0} \equiv N \ , \ N_1 \parallel N_2 \equiv N_2 \parallel N_1 \ , \ (N_1 \parallel N_2) \parallel N_3 \equiv N_1 \parallel (N_2 \parallel N_3)$$

(ALPHA) $N \equiv N' \qquad \text{if } N =_\alpha N'$

(RCOM) $(\nu l_1)(\nu l_2)N \equiv (\nu l_2)(\nu l_1)N$

(EXT) $N_1 \parallel (\nu l)N_2 \equiv (\nu l)(N_1 \parallel N_2) \qquad \text{if } l \notin fn(N_1)$

(ABS) $l ::_\rho C \equiv l ::_\rho (C \mid \mathbf{nil})$

(CLONE) $l ::_\rho C_1 | C_2 \equiv l ::_\rho C_1 \parallel l ::_\rho C_2$

(REC) $l ::_\rho \mathbf{rec}\, X.P \equiv l ::_\rho P[\mathbf{rec}\, X.P/X]$

Reduction Relation:

$$\text{(RED-OUT)} \ \frac{\rho(u) = l' \qquad \mathcal{E}[\![\ t\ ]\!]_\rho = t'}{l ::_\rho \mathbf{out}(t)@u.P \ \parallel \ l' ::_{\rho'} \mathbf{nil} \ \longmapsto \ l ::_\rho P \parallel l' ::_{\rho'} \langle t' \rangle}$$

$$\text{(RED-EVAL)} \ \frac{\rho(u) = l'}{l ::_\rho \mathbf{eval}(P_2)@u.P_1 \ \parallel \ l' ::_{\rho'} \mathbf{nil} \ \longmapsto \ l ::_\rho P_1 \parallel l' ::_{\rho'} P_2}$$

$$\text{(RED-IN)} \ \frac{\rho(u) = l' \qquad match(\mathcal{E}[\![\ T\ ]\!]_\rho, t) = \sigma}{l ::_\rho \mathbf{in}(T)@u.P \ \parallel \ l' ::_{\rho'} \langle t \rangle \ \longmapsto \ l ::_\rho P\sigma \parallel l' ::_{\rho'} \mathbf{nil}}$$

$$\text{(RED-READ)} \ \frac{\rho(u) = l' \qquad match(\mathcal{E}[\![\ T\ ]\!]_\rho, t) = \sigma}{l ::_\rho \mathbf{read}(T)@u.P \ \parallel \ l' ::_{\rho'} \langle t \rangle \ \longmapsto \ l ::_\rho P\sigma \parallel l' ::_{\rho'} \langle t \rangle}$$

$$\text{(RED-NEW)} \ l ::_\rho \mathbf{new}(l').P \ \longmapsto \ (\nu l')(l ::_\rho P \parallel l' ::_{\rho[l'/\texttt{self}]} \mathbf{nil})$$

$$\text{(RED-PAR)} \ \frac{N_1 \ \longmapsto \ N_1'}{N_1 \parallel N_2 \ \longmapsto \ N_1' \parallel N_2}$$

$$\text{(RED-RES)} \ \frac{N \ \longmapsto \ N'}{(\nu l)N \ \longmapsto \ (\nu l)N'}$$

$$\text{(RED-STRUCT)} \ \frac{N \ \equiv \ M \ \longmapsto \ M' \ \equiv \ N'}{N \ \longmapsto \ N'}$$

Table 3
KLAIM Operational Semantics

Formally, *match* is defined by the following rules:

$$match(l, l) = \epsilon \qquad match(!x, l) = [{}^{l}/_{x}]$$

$$match(!X, P) = [{}^{P}/_{X}] \qquad \frac{match(T_1, t_1) = \sigma_1 \qquad match(T_2, t_2) = \sigma_2}{match(\ T_1, T_2\ ,\ t_1, t_2\ ) = \sigma_1 \circ \sigma_2}$$

where we let 'ϵ' to be the empty substitution and '$\circ$' to denote substitutions composition. Here, a substitution σ is a function mapping names and processes into variables; $P\sigma$ denotes the (capture avoiding) application of σ to P. Moreover, we assume that $P\sigma$ yields a process written according to the syntax of Table 2.

The intuition beyond the operational rules of KLAIM is the following. In rule (RED-OUT), the local allocation environment is used both to determine the name of the node where the tuple must be placed and to evaluate the argument tuple. This implies that if the argument tuple contains a field with a process, the corresponding field of the evaluated tuple contains the process resulting from the evaluation of its free variables. Hence, processes in a tuple are transmitted after the interpretation of their free variables through the local allocation environment. This corresponds to having a *static scoping* discipline for the (possibly remote) generation of tuples. A *dynamic linking* strategy is adopted for the **eval** operation, rule (RED-EVAL). In this case the free variables of the spawned process are not interpreted using the local allocation environment: the linking of variables is done at the remote node. Rules (RED-IN) and (RED-READ) require existence of a matching datum in the target node. The tuple is then used to replace the free occurrences of the variables bound by the template in the continuation of the process performing the actions. With action **in**, the matched datum is consumed while with action **read** it is not. Finally, in rule (RED-NEW), the environment of a new node is derived from that of the creating one with the obvious update for the `self` variable. Therefore, the new node inherits all the bindings of the creating node.

2.2 μKLAIM: *micro* KLAIM

The calculus μKLAIM has been derived in [19] from KLAIM by removing allocation environments and the possibility of having pieces of code as tuple fields. [1] Its syntax is given in Table 4. The removal of allocation environments makes it possible to merge together names and variables. Thus, we only assume a countable set $\mathcal{N}$ of *names* $l, l', \ldots, u, \ldots, x, y, \ldots, X, Y, \ldots$. Names provide the abstract counterpart of the set of *communicable* objects and can be used as localities, basic variables

[1] The calculus used in this paper slightly differs from the calculus given in [19]: the differences are the absence of values and expressions (to simplify reasoning) and the use of recursion. These simplifications have been motivated in Remark 2.1.

$N ::= \mathbf{0}$	$l :: C$	$N_1 \parallel N_2$	$(\nu l)N$	$C ::=$ like in Table 2
$t ::= u$	t_1, t_2			$P ::=$ like in Table 2
$T ::= u$	$!\, x$	T_1, T_2		$a ::=$ like in Table 2

Table 4
μKLAIM Syntax

$$(\text{RED-OUT})\quad l :: \mathbf{out}(t)@l'.P \;\parallel\; l' :: \mathbf{nil} \longmapsto l :: P \parallel l' :: \langle t \rangle$$

$$(\text{RED-EVAL})\quad l :: \mathbf{eval}(P_2)@l'.P_1 \;\parallel\; l' :: \mathbf{nil} \longmapsto l :: P_1 \parallel l' :: P_2$$

$$(\text{RED-IN})\quad \frac{match(T, t) = \sigma}{l :: \mathbf{in}(T)@l'.P \;\parallel\; l' :: \langle t \rangle \longmapsto l :: P\sigma \parallel l' :: \mathbf{nil}}$$

$$(\text{RED-READ})\quad \frac{match(T, t) = \sigma}{l :: \mathbf{read}(T)@l'.P \;\parallel\; l' :: \langle t \rangle \longmapsto l :: P\sigma \parallel l' :: \langle t \rangle}$$

$$(\text{RED-NEW})\quad l :: \mathbf{new}(l').P \longmapsto (\nu l')(l :: P \parallel l' :: \mathbf{nil})$$

Table 5
μKLAIM Distinctive Reduction Rules

or process variables: we do not need to distinguish between these three kinds of objects anymore. Like before, we prefer letters $l, l', \ldots$ when we want to stress the use of a name as a locality, $x, y, \ldots$ when we want to stress the use of a name as a basic variable, and $X, Y, \ldots$ when we want to stress the use of a name as a process variable. We will use u for basic variables and localities.

Notice that μKLAIM can be considered as the largest sub-calculus of KLAIM where tuples do not contain any process, allocation environments are empty and all processes are closed. These modifications sensibly simplifies the operational semantics of the language. The structural congruence is readily adapted from Table 3; the key laws to define the reduction relation are given in Table 5. Notice that now tuples/templates evaluation function is useless and substitutions are (standard) functions of names. Hence, the definition of function *match* is given by the following laws:

$$match(l, l) = \epsilon \qquad \frac{match(T_1, t_1) = \sigma_1 \quad match(T_2, t_2) = \sigma_2}{match(\, T_1, T_2 \,,\, t_1, t_2 \,) = \sigma_1 \circ \sigma_2}$$

$$match(!x, l) = [{}^{l}/_{x}]$$

N ::= like in Table 4	a ::= $\mathbf{in}(T)@u \mid \mathbf{out}(t)@u \mid \mathbf{eval}(P)@u \mid \mathbf{new}(l)$
C ::= like in Table 4	t ::= u
P ::= like in Table 4	T ::= $u \mid !x$

Table 6
cKLAIM Syntax

2.3 cKLAIM*: core* KLAIM

The calculus cKLAIM has been introduced in [13] by eliminating from μKLAIM action **read** and by only considering monadic communications (i.e. tuples and templates containing only one field). The formal syntax of cKLAIM is given in Table 6. Notice that cKLAIM is a sub-calculus of μKLAIM and thus it inherits from μKLAIM the operational semantics.

2.4 lcKLAIM*: local core* KLAIM

lcKLAIM is the version of cKLAIM where actions **out** and **in** can be only performed locally, i.e. the only remote primitive is action **eval** (this is the principle underlying the language Dπ [20]). The syntax of the new calculus can be derived from the syntax of cKLAIM (see Table 6) by using the following production for process actions:

$$a \quad ::= \quad \mathbf{in}(T) \mid \mathbf{out}(t) \mid \mathbf{eval}(P)@u \mid \mathbf{new}(l)$$

We want to remark that lcKLAIM is a sub-calculus of cKLAIM: indeed, it is the largest sub-calculus of cKLAIM closed under the predicate $\rightsquigarrow$, defined as

$$N \rightsquigarrow \triangleq N = \mathbf{0} \quad \vee \quad (N = (\nu l)N' \ \wedge \ N' \rightsquigarrow) \quad \vee$$
$$(N = N_1 \parallel N_2 \ \wedge \ N_1 \rightsquigarrow \ \wedge N_2 \rightsquigarrow) \quad \vee \quad (N = l :: C \ \wedge \ C \rightsquigarrow_l)$$

$$C \rightsquigarrow_l \triangleq C = \langle l' \rangle \ \vee \ (C = P \ \wedge \ P \rightsquigarrow_l) \ \vee \ (C = C_1 | C_2 \ \wedge \ C_1 \rightsquigarrow_l \ \wedge C_2 \rightsquigarrow_l)$$

$$P \rightsquigarrow_u \triangleq (P = \mathbf{nil}, X) \quad \vee \quad (P = \mathbf{eval}(Q)@v.R \ \wedge \ Q \rightsquigarrow_v \ \wedge R \rightsquigarrow_u) \quad \vee$$
$$(P = P_1 | P_2 \ \wedge \ P_1 \rightsquigarrow_u \ \wedge P_2 \rightsquigarrow_u) \quad \vee$$
$$(P = \mathbf{in}(T)@u.Q, \mathbf{out}(t)@u.Q, \mathbf{new}(l).Q, \mathbf{rec}\, X.Q \ \wedge \ Q \rightsquigarrow_u)$$

The only relevant cases are those for prefixes **in**/**out**/**eval**. They ensure that actions **in** and **out** only specify as target node the node where the action is executed (i.e. the u decorating $\rightsquigarrow_u$).

The operational semantics of lcKlaim is obtained by replacing rules (Red-Out) and (Red-In) of Table 5 with the following ones:

$$\text{(Red-Out)} \qquad l :: \mathbf{out}(l').P \longmapsto l :: P \mid \langle l' \rangle$$

$$\text{(Red-In)} \qquad l :: \mathbf{in}(T).P \mid \langle l' \rangle \longmapsto l :: P\sigma \qquad \text{if } match(T, l') = \sigma$$

2.5 Observational Semantics

A *net context* $\mathcal{C}[\cdot]$ is a net with an occurrence of a hole $[\cdot]$ to be filled with any net. [2] Formally,

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

We now give the main equivalences we shall work with throughout this paper, namely *barbed bisimilarity* and *reduction barbed congruence*. To this aim, we start by defining an intuitive notion of observable, or *barb*.

Definition 2.3 (Barbs) *Predicate $N \downarrow l$ holds true if and only if $N \equiv (\nu \widetilde{l})(N' \parallel l ::_\rho \langle t \rangle)$ for some $\widetilde{l}$, N' and t such that $l \notin \widetilde{l}$. Predicate $N \Downarrow l$ holds true if and only if $N \longmapsto^* N'$, for some N' such that $N' \downarrow l$.*

Definition 2.4 *Let $\mathfrak{R}$ be a binary relation between nets. $\mathfrak{R}$ is said*

- *barb preserving, if $N \mathfrak{R} M$ and $N \downarrow l$ imply $M \Downarrow l$*
- *reduction closed, if $N \mathfrak{R} M$ and $N \longmapsto N'$ imply $M \longmapsto^* M'$ and $N' \mathfrak{R} M'$, for some M'*
- *context closed, if $N \mathfrak{R} M$ implies $\mathcal{C}[N] \mathfrak{R} \mathcal{C}[M]$ for every net context $\mathcal{C}[\cdot]$.*

Definition 2.5 (Barbed Bisimilarity) *A symmetric relation $\mathfrak{R}$ between nets is a* barbed bisimulation *if it is barb preserving and reduction closed.* Barbed bisimilarity, $\dot{\cong}$, *is the largest barbed bisimulation.*

Definition 2.6 (Reduction Barbed Congruence) Reduction barbed congruence, $\cong$, *is the largest symmetric, barb preserving, reduction and context closed relation between nets.*

[2] In the case of Klaim, we implicitly assume that the hole of a context $\mathcal{C}[\cdot]$ can be filled only with those nets N such that the resulting net $\mathcal{C}[N]$ is well-formed, i.e. allocation environments in clones of the same node coincide.

In this section, we set up the technical background needed for establishing the properties enjoyed by the encodings. We start by presenting the necessary notions and notations for μKLAIM; the corresponding ones for cKLAIM and lcKLAIM are strictly related to them and are sketched at the end.

We start by introducing a *labelled transition system* (LTS, for short) that describes the evolution of a net and provides information about the performed actions [13]. The LTS is given in Table 7 and uses labels as generated by the following BNF (we use $I ::= \mathbf{nil} \mid \langle t \rangle$ to denote *inert* node components):

$$\chi ::= \tau \mid (\nu \widetilde{l})\, I @ l \qquad\qquad \alpha ::= \chi \mid \triangleright\, l \mid t \triangleleft l_1$$

Let us now briefly comment on some rules of the LTS; most of them are adapted from the π-calculus [30]. Rule (LTS-EXISTS) signals existence of nodes (label $\mathbf{nil}\ @\ l$) or of data (label $\langle t \rangle\ @\ l$). Rules (LTS-OUT) and (LTS-EVAL) express the intention of spawning a component and require the existence of the target node to complete successfully (rule (LTS-SEND)). Similarly, rules (LTS-IN) (given in an early style) and (LTS-READ) express the intention of performing an input; this input is actually performed (rule (LTS-COMM)) only if the chosen datum is present in the target node. Notice that, in the right hand side of these rules, existence of the node target of the action can be assumed: indeed, if l provides datum $\langle t \rangle$, this implies that l does exist. Rule (LTS-OPEN) signals extrusion of bound names; as in some presentation of π-calculus (see, e.g., [27]), this rule is used to investigate the capability of processes to export bound names, rather than to extend the scope of bound names. To this last aim, law (EXT) is used; in fact, in rule (LTS-COMM) labels do not carry any restriction on names, whose scope must have been previously extended. Rules (LTS-RES), (LTS-PAR) and (LTS-STRUCT) are standard.

Notation 2.7 We shall write $N \xrightarrow{\alpha}$ to mean that there exists a net N' such that $N \xrightarrow{\alpha} N'$. Alternatively, we could say that N can perform an 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 $\Rightarrow$ to stand for $\xrightarrow{\tau}{}^*$ and $\stackrel{\alpha}{\Rightarrow}$ to stand for $\Rightarrow\xrightarrow{\alpha}\Rightarrow$. Notation $\stackrel{\hat{\alpha}}{\Rightarrow}$ stands for $\Rightarrow$, if $\alpha = \tau$, and for $\stackrel{\alpha}{\Rightarrow}$, otherwise; similarly, $\xrightarrow{\hat{\alpha}}$ stands for $\xrightarrow{\alpha}$ if $\alpha \neq \tau$, and for either $\xrightarrow{\tau}$ or the identity, otherwise.

We now present some useful properties of the LTS that substantiate its use throughout the paper. We start with a simple Proposition that relates the labels of the LTS with the syntax of the net performing the labelled action.

(LTS-Out)

$$l :: \mathbf{out}(t)@l'.P \xrightarrow{\triangleright\, l'} l :: P \parallel l' :: \langle t \rangle$$

(LTS-Eval)

$$l :: \mathbf{eval}(Q)@l'.P \xrightarrow{\triangleright\, l'} l :: P \parallel l' :: Q$$

(LTS-In)

$$\frac{match(T, t) = \sigma}{l :: \mathbf{in}(T)@l'.P \xrightarrow{t \,\triangleleft\, l'} l :: P\sigma \parallel l' :: \mathbf{nil}}$$

(LTS-Read)

$$\frac{match(T, t) = \sigma}{l :: \mathbf{read}(T)@l'.P \xrightarrow{t \,\triangleleft\, l'} l :: P\sigma \parallel l' :: \langle t \rangle}$$

(LTS-New)

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

(LTS-Exists)

$$l :: I \xrightarrow{I \,@\, l} l :: \mathbf{nil}$$

(LTS-Comm)

$$\frac{N_1 \xrightarrow{t \,\triangleleft\, l'} N_1' \qquad N_2 \xrightarrow{\langle t \rangle \,@\, l'} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

(LTS-Send)

$$\frac{N_1 \xrightarrow{\mathbf{nil} \,@\, l} N_1' \qquad N_2 \xrightarrow{\triangleright\, l} N_2'}{N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2'}$$

(LTS-Res)

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

(LTS-Open)

$$\frac{N \xrightarrow{(\nu \widetilde{l})\ \langle t \rangle \,@\, l'} N' \qquad l \in fn(t) - \{\widetilde{l}, l'\}}{(\nu l)N \xrightarrow{(\nu \widetilde{l}, l)\ \langle t \rangle \,@\, l'} N'}$$

(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)

$$\frac{N \equiv M \xrightarrow{\alpha} M' \equiv N'}{N \xrightarrow{\alpha} N'}$$

Table 7
μKlaim Labelled Transition System (LTS)

Proposition 2.8 *The following facts hold:*

(1) $N \xrightarrow{\mathbf{nil} \,@\, l} N'$ *if and only if* $N \equiv N'' \parallel l :: \mathbf{nil}$*; moreover,* $N'' \equiv N' \equiv N$*.*
(2) $N \xrightarrow{(\nu \widetilde{l})\ \langle t \rangle \,@\, l} N'$ *if and only if* $N \equiv (\nu \widetilde{l})(N'' \parallel l :: \langle t \rangle)$ *for* $l \notin \widetilde{l}$*; moreover,* $N' \equiv N'' \parallel l :: \mathbf{nil}$*.*

Then, we can use the LTS to describe the possible evolutions of a net put in a context. This result will enable the development of the proofs of this paper.

Proposition 2.9 $\mathcal{C}[N] \xrightarrow{\alpha} \overline{N}$ *if and only if one of the following conditions hold:*

(1) $N \xrightarrow{\alpha} N'$ *with* $n(\alpha) \cap bn(\mathcal{C}[\cdot]) = \emptyset$
(2) $\mathcal{C}[\mathbf{0}] \xrightarrow{\alpha} \mathcal{C}'[\mathbf{0}]$

(3) $N \xrightarrow{\alpha'} N'$ *with* $\alpha = (\nu l)\alpha'$, $\mathcal{C}[\cdot] \triangleq \mathcal{C}_1[(\nu l)\mathcal{C}_2[\cdot]]$ *and* $fn(\alpha) \cap bn(\mathcal{C}_1[\cdot], \mathcal{C}_2[\cdot]) = \emptyset$

(4) $\mathcal{C}[\cdot] \triangleq \mathcal{C}_1[\mathcal{C}_2[\cdot] \parallel H]$ *with* $H \xrightarrow{\textbf{nil} @ l} H'$, $N \xrightarrow{\triangleright l} N'$ *and* $l \notin bn(\mathcal{C}_2[\cdot])$

(5) $\mathcal{C}[\cdot] \triangleq \mathcal{C}_1[\mathcal{C}_2[\cdot] \parallel H]$ *with* $H \xrightarrow{\triangleright l} H'$, $N \xrightarrow{\textbf{nil} @ l} N'$ *and* $l \notin bn(\mathcal{C}_2[\cdot])$

(6) $\mathcal{C}[\cdot] \equiv \mathcal{C}_1[\mathcal{C}_2[\cdot] \parallel H]$ *with* $H \xrightarrow{\langle t \rangle @ l} H'$, $N \xrightarrow{t \blacktriangleleft l} N'$ *and* $\{l, t\} \cap bn(\mathcal{C}_2[\cdot]) = \emptyset$

(7) $\mathcal{C}[\cdot] \equiv \mathcal{C}_1[\mathcal{C}_2[\cdot] \parallel H]$ *with* $H \xrightarrow{t \blacktriangleleft l} H'$, $N \xrightarrow{(\widetilde{\nu l})\ \langle t \rangle @ l} N'$ *and* $\{l, t\} \notin bn(\mathcal{C}_2[\cdot])$.

Moreover, the resulting net $\overline{N}$ *is, respectively, structurally equivalent to* $\mathcal{C}[N']$, *or* $\mathcal{C}'[N]$, *or* $\mathcal{C}_1[\mathcal{C}_2[N']]$, *or* $\mathcal{C}[N']$, *or* $\mathcal{C}_1[\mathcal{C}_2[N] \parallel H']$, *or* $\mathcal{C}_1[\mathcal{C}_2[N'] \parallel H']$, *or* $\mathcal{C}_1[(\widetilde{\nu l})\mathcal{C}_2[N' \parallel H']]$. *Finally,* $\alpha = \tau$ *in cases 4., 5., 6., and 7. .*

Proof: The "if" part is trivial, by using the LTS of Table 7 and by observing that $M \xrightarrow{\alpha} M'$ with $n(\alpha) \cap bn(\mathcal{D}[\cdot]) = \emptyset$ implies $\mathcal{D}[M] \xrightarrow{\alpha} \mathcal{D}[M']$. The "only if" part is proved by induction on the length of the inference of $\xrightarrow{\alpha}$. In the base case (length 1), it must be $\mathcal{C}[\cdot] \triangleq [\,\cdot\,]$; hence, obviously $\mathcal{C}[N] \triangleq N \xrightarrow{\alpha} N' \triangleq \mathcal{C}[N']$ and we trivially fall in case (1) of this Lemma. For the inductive step, we reason by case analysis on the last rule applied in the inference; the proof is long and quite standard. Thus, we only sketch here the most delicate case; for full details, see [18]. Let us assume that the last rule used to infer the transition has been (LTS-Struct); thus,

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

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

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

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

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

- $K \xrightarrow{\alpha} K'$ and $\overline{N} \equiv \mathcal{D}[N] \parallel K'$. In this case, we are trivially in case (2) of this Lemma.

- $\mathcal{D}[N] \xrightarrow{\alpha} \overline{N}'$ and $\overline{N} \equiv \overline{N}' \parallel K$. In this case, we use the structural induction.
- If $\alpha = \tau$ then other four cases are possible:
  - $\mathcal{D}[N] \xrightarrow{\triangleright\, l} \overline{N}'$, $K \xrightarrow{\mathbf{nil}\ @\ l} K$ and $\overline{N} \equiv \overline{N}' \parallel K$. By structural induction, it can be that either $N \xrightarrow{\triangleright\, l} N'$, or $\mathcal{D}[\mathbf{0}] \xrightarrow{\triangleright\, l} \mathcal{D}'[\mathbf{0}]$. In both cases is easy to conclude.
  - $\mathcal{D}[N] \xrightarrow{(\nu \tilde{l})\ \langle t \rangle\ @\ l} \overline{N}'$, $K \xrightarrow{t \,\triangleleft\, l} K'$ and $\overline{N} \equiv (\widetilde{\nu l})(\overline{N}' \parallel K')$. This case is similar to the previous one.
  - $\mathcal{D}[N] \xrightarrow{\mathbf{nil}\ @\ l} \mathcal{D}[N]$, $K \xrightarrow{\triangleright\, l} K'$ and $\overline{N} \equiv \mathcal{D}[N] \parallel K'$. By structural induction, it can be one of the first two cases of this Lemma and we can easily conclude.
  - $\mathcal{D}[N] \xrightarrow{t \,\triangleleft\, l} \overline{N}'$, $K \xrightarrow{(\nu \tilde{l})\ \langle t \rangle\ @\ l} K'$ and $\overline{N} \equiv (\widetilde{\nu l})(\overline{N}' \parallel K')$. This case is similar to the previous one. □

By exploiting this result, it is easy to prove that the LTS we have just defined is *sound* w.r.t. the semantics of the calculus.

Proposition 2.10 (Soundness of the LTS) *$N \longmapsto N'$ if and only if $N \xrightarrow{\tau} N'$.*

Because of this result we shall regularly mix the use of reductions and of τ-steps, and use one in place of the other interchangeably.

As we already said in the Introduction, we shall assess the quality of our encodings by using a notion of *translated barbed congruence*. Once fixed an encoding $enc(\cdot)$ from a certain language $\mathcal{L}$ into μKLAIM, this equivalence is defined like barbed congruence but it only consider those contexts that are the encoding (via *enc*) of a source one. By following [5], we shall denote this barbed congruence as $\cong^{\mathtt{tr}}_{\mu K}$ (because the contexts considered are always *tr*anslated, via *enc*). However, in the proofs, it will be convenient to keep track of the number of τ-steps a net requires to simulate the other while establishing barbed congruence. This gives rise to a preorder on nets that we call *barbed expansion*. Recall from Notation 2.7 that $N \xrightarrow{\hat{\tau}} N'$ stands for either $N \equiv N'$ or $N \xrightarrow{\tau} N'$.

Definition 2.11 (Barbed Expansion Preorder) *A preorder $\mathfrak{R}$ between μ*KLAIM *nets is a* barbed expansion *if for each $N_1 \mathfrak{R} N_2$ it holds that:*

(1) if $N_1 \downarrow l$ then $N_2 \Downarrow l$;
(2) if $N_2 \downarrow l$ then $N_1 \downarrow l$;
(3) if $N_1 \xrightarrow{\tau} N_1'$ then $N_2 \stackrel{\tau}{\Rightarrow} N_2'$ and $N_1' \mathfrak{R} N_2'$, for some N_2';
(4) if $N_2 \xrightarrow{\tau} N_2'$ then $N_1 \xrightarrow{\hat{\tau}} N_1'$ and $N_1' \mathfrak{R} N_2'$, for some N_1';
(5) $\mathcal{C}[N_1]\ \mathfrak{R}\ \mathcal{C}[N_2]$, for every context $\mathcal{C}[\cdot]$.

The expansion preorder, *$\lesssim_{\mu K}$, is the largest barbed expansion (when notationally useful, we write $N \lesssim_{\mu K} M$ as $M \gtrsim_{\mu K} N$).*

Like barbed congruence, barbed expansion can be defined by requiring closure only under a subset of language contexts. In particular, once fixed an encoding $enc(\cdot)$ from a certain language $\mathcal{L}$ into μKLAIM, we define $\lesssim^{\mathrm{tr}}_{\mu K}$, the *translated* barbed expansion, to be the largest relation defined like $\lesssim_{\mu K}$, but where context closure only consider those contexts $\mathcal{C}[\cdot]$ such that $\mathcal{C}[\cdot] = enc(\mathcal{D}[\cdot])$ and $\mathcal{D}[\cdot]$ is an $\mathcal{L}$-context. We let $enc(\mathcal{D}[\cdot])$ be defined as a standard net encoding that replaces $[\cdot]$ with $[\cdot]$. We now establish an ordering among the relations introduced so far.

Proposition 2.12 $\equiv \subset \lesssim_{\mu K} \subset \cong_{\mu K}$ *and* $\equiv \subset \lesssim^{\mathrm{tr}}_{\mu K} \subset \cong^{\mathrm{tr}}_{\mu K}$.

Proof: We just prove the first statement; the second one can be proved similarly. The inclusion $\equiv \subset \lesssim_{\mu K}$ is simple: proving '$\subseteq$' is straightforward, while the first four statements of Proposition 2.16 can be used to prove that the reverse inclusion does not hold. The inclusion $\lesssim_{\mu K} \subset \cong_{\mu K}$ holds by definition. □

In what follows, we shall use some well-established proof techniques, namely *up-to expansion techniques*. We say that $\mathfrak{R}$ is a barbed congruence up-to $\lesssim_{\mu K}$ if it is defined like in Definition 2.5 but reduction and context closure are weakened and consider $\gtrsim_{\mu K}\mathfrak{R}\lesssim_{\mu K}$ (instead of $\mathfrak{R}$) in the closure. The translated versions of barbed congruence and expansion are modified similarly. Formally, we have the following definitions.

Definition 2.13 (Barbed Congruence up-to $\lesssim_{\mu K}$) *A symmetric relation between μ*KLAIM *nets $\mathfrak{R}$ is a* barbed congruence up-to $\lesssim_{\mu K}$ *if, whenever $N_1\ \mathfrak{R}\ N_2$, it holds that:*

- *if $N_1 \downarrow l$ then $N_2 \Downarrow l$;*
- *if $N_1 \xrightarrow{\tau} N_1'$ then there exists N_2' such that $N_2 \Rightarrow N_2'$ and $N_1' \gtrsim_{\mu K}\mathfrak{R}\lesssim_{\mu K} N_2'$;*
- *for every context $\mathcal{C}[\cdot]$, it holds that $\mathcal{C}[N_1] \gtrsim_{\mu K}\mathfrak{R}\lesssim_{\mu K} \mathcal{C}[N_2]$.*

Definition 2.14 (Translated Barbed Congruence up-to $\lesssim^{\mathrm{tr}}_{\mu K}$) *A symmetric relation between μ*KLAIM *nets $\mathfrak{R}$ is a* translated barbed congruence up-to $\lesssim^{\mathrm{tr}}_{\mu K}$ *if, whenever $N_1\ \mathfrak{R}\ N_2$, it holds that:*

- *if $N_1 \downarrow l$ then $N_2 \Downarrow l$;*
- *if $N_1 \xrightarrow{\tau} N_1'$ then there exists N_2' such that $N_2 \Rightarrow N_2'$ and $N_1' \gtrsim^{\mathrm{tr}}_{\mu K}\mathfrak{R}\lesssim^{\mathrm{tr}}_{\mu K} N_2'$;*
- *$\mathcal{C}[N_1] \gtrsim^{\mathrm{tr}}_{\mu K}\mathfrak{R}\lesssim^{\mathrm{tr}}_{\mu K} \mathcal{C}[N_2]$, for every translated context $\mathcal{C}[\cdot]$.*

Proposition 2.15 (Up-to Techniques) *The following facts hold:*

(1) if $\mathfrak{R}$ is a barbed congruence up-to $\lesssim_{\mu K}$, then $\mathfrak{R} \subseteq \cong_{\mu K}$.
(2) if $\mathfrak{R}$ is a translated barbed congruence up-to $\lesssim^{\mathrm{tr}}_{\mu K}$, then $\mathfrak{R} \subseteq \cong^{\mathrm{tr}}_{\mu K}$.

Proof: The proofs of the two claims are similar; we just show the first one. It suffices to prove that $\mathfrak{I} \triangleq \{(N, M) : N\gtrsim_{\mu K}\mathfrak{R}\lesssim_{\mu K}M\}$ is barb preserving, reduction closed and closed under translated contexts. We consider $N\gtrsim_{\mu K}N_1\ \mathfrak{R}\ M_1\lesssim_{\mu K}M$. Let

$N \xrightarrow{\tau} N'$. Then, by hypothesis, $N_1 \xrightarrow{\hat{\tau}} N_2$ and $N' \gtrsim_{\mu K} N_2$. Now, if $N_1 \equiv N_2$, we can state that $N' \gtrsim_{\mu K} N_1$; hence, $M \Rightarrow M$ and $N' \; \Im \; M$. On the other hand, if $N_1 \xrightarrow{\tau} N_2$ then $M_1 \stackrel{\hat{\tau}}{\Rightarrow} M_2$ and $N_2 \gtrsim_{\mu K} \Re \lesssim_{\mu K} M_2$. Then, $M \stackrel{\hat{\tau}}{\Rightarrow} M'$ and $M_2 \lesssim_{\mu K} M'$; hence, by transitivity of $\lesssim_{\mu K}$ (that can be easily proved), we obtain $N' \; \Re \; M'$, as required. Now, let $N \downarrow l$; then, $N_1 \downarrow l$. Then, $M_1 \Downarrow l$, i.e. $M_1 \Rightarrow M_2 \downarrow l$. Now, $M \Rightarrow M'$ and $M_2 \lesssim_{\mu K} M'$; thus, $M' \Downarrow l$ and, hence, $M \Downarrow l$, as required. Finally, context closure holds by definition. □

We now give some simple laws that greatly simplify our proofs.

Proposition 2.16 *The following facts hold:*

(1) $(\nu l')(l :: P\sigma \parallel l' :: \mathbf{nil}) \lesssim_{\mu K} (\nu l')(l :: \mathbf{in}(T)@l'.P \parallel l' :: \langle t \rangle)$ *whenever* $match(T, t) = \sigma$
(2) $l :: P \parallel l' :: \langle t \rangle \lesssim_{\mu K} l :: \mathbf{out}(t)@l'.P \parallel l' :: \mathbf{nil}$
(3) $l :: P \parallel l' :: Q \lesssim_{\mu K} l :: \mathbf{eval}(Q)@l'.P \parallel l' :: \mathbf{nil}$
(4) $(\nu l')(l :: P \parallel l' :: \mathbf{nil}) \lesssim_{\mu K} l :: \mathbf{new}(l').P$
(5) $(\nu l)(l :: I) \lesssim_{\mu K} \mathbf{0} \lesssim_{\mu K} (\nu l)(l :: I)$.

Technicalities for cKlaim and lcKlaim. Most of the theory presented for μKlaim can be easily adapted to cKlaim and lcKlaim. In particular, an LTS for cKlaim can be obtained from the rules in Table 7 by removing the rule for action **read** and by only considering monadic tuples/templates. The LTS for lcKlaim is obtained by replacing the rules of cKlaim for actions **out** and **in** with the following ones:

$$l :: \mathbf{out}(l').P \xrightarrow{\tau} l :: P \mid \langle l' \rangle \qquad \frac{match(T, l') = \sigma}{l :: \mathbf{in}(T).P \xrightarrow{l' \triangleleft l} l :: P\sigma}$$

Then, we denote with $\cong_{cK}$ the restriction of $\cong_{\mu K}$ to cKlaim nets; clearly, $\cong_{cK} \subseteq \cong_{\mu K}$. Relations $\lesssim_{cK}$, $\cong^{\mathrm{tr}}_{cK}$ and $\cong^{\mathrm{tr}}_{cK}$ are defined similarly. Finally, we define similar relations $\lesssim_{lcK}$ and $\cong_{lcK}$ for lcKlaim. Clearly, all the properties stated and proved in this section for μKlaim can be faithfully rephrased to deal with the sub-relations containing only cKlaim or lcKlaim nets.

3 Klaim vs μKlaim

Intuitions. There are two differences between Klaim and μKlaim: presence/absence of allocation environments and presence/absence of higher-order communications. Intuitively, allocation environments are translated into tuples of the TS allocated at a reserved locality env. If the allocation environment ρ of l maps x to l', then a tuple $\langle l, x, l' \rangle$ is stored at env. Hence, when performing an action **out**/**in**/**read**, all the (originally) free variables occurring in the tuple/template

must be translated according to the current allocation environment. This is made possible by adding a sequence of actions **read** to properly translate the free variables. Notice, however, that a renaming of the free variables with fresh ones is necessary not to capture occurrences of the same variables within the scope of prefixed actions **eval** (this is necessary to correctly implement the dynamic binding of these variables). Informally, the KLAIM node

$$l_1 ::_{\rho_1} P$$

with

$$P \triangleq \mathbf{out}(x, l')@y.\mathbf{eval}(\mathbf{out}(x, l')@y)@x \tag{1}$$

and ρ_1 such that $\rho_1(x) = l_1$ and $\rho_1(y) = l_2$, is translated into the μKLAIM net

$$l_1 :: P' \quad \| \quad \mathsf{env} :: \langle l, x, l_1\rangle \mid \langle l, y, l_2\rangle \mid \ldots$$

where

$$\begin{aligned} P' \triangleq\ & \mathbf{read}(l, x, !x')@\mathsf{env}.\mathbf{read}(l, y, !y')@\mathsf{env}. \\ & \mathbf{out}(x', l')@y'.\mathbf{eval}(\mathbf{out}(x, l')@y)@x' \end{aligned} \tag{2}$$

Since the name binding discipline implemented for actions **out** is static, the theory developed for higher-order π-calculus [29] by means of *triggers* can be smoothly integrated to the present setting. In *loc.cit.*, a HOπ-calculus process

$$\bar{a}\langle p\rangle \quad | \quad a(X).X$$

is translated to

$$(\nu c)(\bar{a}\langle c\rangle \mid !c().\acute{p}) \quad | \quad a(x).\bar{x}\langle\rangle$$

where $\bar{a}\langle p\rangle$ sends process p on channel a and $\acute{p}$ is the translation of p (for a more precise syntax and semantics of π-calculus see Section 6.1). The idea of this encoding is to assign a fresh pointer c to p and distribute it in place of p. Such pointer is then used by the interested processes to activate as many copies of p as needed. This idea can be faithfully adapted to KLAIM. For example, the net

$$l_1 ::_{\rho_1} \mathbf{out}(P)@l_1 \ \| \ l_2 ::_{\rho_2} \mathbf{in}(!X)@l_1.X$$

where P is defined like in (1), is translated into

$$l_1 :: \mathbf{new}(l).\mathbf{eval}(P_l)@l.\mathbf{out}(l)@l_1 \quad \| \quad l_2 :: \mathbf{in}(!x)@l_1.\mathbf{out}(l_2)@x \quad \| \quad \mathsf{env} :: \ldots$$

where

$$P_l \triangleq \mathbf{rec}\, X.\mathbf{in}(!z)@l.(\, X \mid \mathbf{eval}(P')@z\,)$$

and P' is defined like in (2).

As this intuitive discussion should have clarified, name translation and handling of higher-order data are compatible issues. In particular, the full abstraction result of

$$(\!|\mathbf{0}|\!) \triangleq \mathtt{env} :: \langle\rangle \qquad (\!|N_1 \parallel N_2|\!) \triangleq (\!|N_1|\!) \parallel (\!|N_2|\!)$$

$$(\!|(\nu l)N|\!) \triangleq (\nu l)(\!|N|\!) \qquad (\!|l ::_\rho C|\!) \triangleq l :: (\!|C|\!)_{l;fv(C)} \parallel \mathtt{env} :: \langle l, l\rangle \mid \prod_{(x,l')\in\rho}^{x\neq \mathtt{self}} \langle l, x, l'\rangle$$

where env is a reserved name.

Table 8
Encoding KLAIM into μKLAIM: Nets

[29] can be established in our framework as well. Nevertheless, a formal presentation of the complete encoding turns out to be notationally overcomplicated. Thus, from now on, we only consider the first-order fragment of KLAIM, i.e. those KLAIM nets that do not contain processes in tuple fields.

Formal development. We now formalise the way in which we can simulate in μKLAIM the translation via allocation environments of free variables to locality names. This is done by the encoding presented in Table 8, where env is a reserved name.

As already said, env's TS collects tuples of the form $\langle l, x, l'\rangle$ to properly record the associations in l's allocation environment. Moreover, node env also contains another kind of tuples, i.e. pairs $\langle l', l\rangle$ stating that the allocation environment of l' coincides with l's one, except for the `self` entry. This is useful when l' is a node created by l. Indeed, we do not duplicate the allocation environment of l in env for l', but we just put a "link" to the original environment; we shall say that l is an *alias* for l'. Clearly, this solution imposes the special handling of variable `self`, that is not implemented as the other entries of an allocation environment but is automatically resolved by the encoding (see the second case for the encoding of action **eval** and the side conditions $(**)$ and $(***)$ for actions **out**, **in** and **read**). Moreover, if l created l' that, in turn, created l'', then env contains the tuples $\langle l', l\rangle$ and $\langle l'', l\rangle$ (see the encoding of action **new**). This is necessary because the allocation environment of l' is, in fact, the environment of l. Thus, when performing an action **out**/**in**/**read**, the translation of the (originally) free variables must be preceded by an action **read** that retrieves the link to the proper allocation environment.

Notice that, when a locality l is present in N, its allocation environment is explicitly stored in env and l is clearly linked to itself (i.e., the tuple $\langle l, l\rangle$ is stored in env). Notice also that, by definition of $(\!|\mathbf{0}|\!)$, the tuple space of env is never empty. This will turn out to be fundamental in order to obtain a fully abstraction result. Moreover, notice that structurally equivalent nets (like **0** and **0** $\parallel$ **0**) may have different encodings. Nevertheless, this is not a problem, since we work with translated barbed congruence, that ignores this fact.

$$\langle\!\langle\ \langle t\rangle\ \rangle\!\rangle_{u;V} \triangleq \langle t\rangle$$

$$\langle\!\langle C_1|C_2\rangle\!\rangle_{u;V} \triangleq \langle\!\langle C_1\rangle\!\rangle_{u;V} \mid \langle\!\langle C_2\rangle\!\rangle_{u;V}$$

$$\langle\!\langle \mathbf{nil}\rangle\!\rangle_{u;V} \triangleq \mathbf{nil}$$

$$\langle\!\langle X\rangle\!\rangle_{u;V} \triangleq X$$

$$\langle\!\langle \mathbf{rec}\, X.P\rangle\!\rangle_{u;V} \triangleq \mathbf{rec}\, X.\langle\!\langle P\rangle\!\rangle_{u;V}$$

$$\langle\!\langle \mathbf{new}(l).P\rangle\!\rangle_{u;V} \triangleq \mathbf{new}(l).\mathbf{read}(u, !y)@\mathtt{env}.\mathbf{out}(l, y)@\mathtt{env}.\langle\!\langle P\rangle\!\rangle_{u;V} \quad \text{where } y \text{ is fresh}$$

$$\langle\!\langle \mathbf{eval}(Q)@v.P\rangle\!\rangle_{u;V} \triangleq \begin{cases} \mathbf{eval}(\langle\!\langle Q\rangle\!\rangle_{v;V})@v.\langle\!\langle P\rangle\!\rangle_{u;V} & \text{if } v \in \mathcal{L} \\ \mathbf{eval}(\langle\!\langle Q\rangle\!\rangle_{u;V})@u.\langle\!\langle P\rangle\!\rangle_{u;V} & \text{if } v = \mathtt{self} \\ \mathbf{read}(u, !y)@\mathtt{env}.\mathbf{read}(y, v, !z)@\mathtt{env}. & \text{if } (*) \\ \quad \mathbf{eval}(\langle\!\langle Q\rangle\!\rangle_{z;V})@z.\langle\!\langle P\rangle\!\rangle_{u;V} & \end{cases}$$

$$\begin{aligned} \langle\!\langle \mathbf{out}(t)@v.P\rangle\!\rangle_{u;V} \triangleq\ & \mathbf{read}(u, !y)@\mathtt{env}.\mathbf{read}(y, x_1, !y_1)@\mathtt{env}.\cdots \quad \text{where } (**) \\ & \cdots.\mathbf{read}(y, x_n, !y_n)@\mathtt{env}.\mathbf{out}(t')@v'.\langle\!\langle P\rangle\!\rangle_{u;V} \end{aligned}$$

$$\begin{aligned} \langle\!\langle \mathbf{in}(T)@v.P\rangle\!\rangle_{u;V} \triangleq\ & \mathbf{read}(u, !y)@\mathtt{env}.\mathbf{read}(y, x_1, !y_1)@\mathtt{env}.\cdots \quad \text{where } (***) \\ & \cdots.\mathbf{read}(y, x_n, !y_n)@\mathtt{env}.\mathbf{in}(T')@v'.\langle\!\langle P\rangle\!\rangle_{u;V} \end{aligned}$$

$$\begin{aligned} \langle\!\langle \mathbf{read}(T)@v.P\rangle\!\rangle_{u;V} \triangleq\ & \mathbf{read}(u, !y)@\mathtt{env}.\mathbf{read}(y, x_1, !y_1)@\mathtt{env}.\cdots \quad \text{where } (***) \\ & \cdots.\mathbf{read}(y, x_n, !y_n)@\mathtt{env}.\mathbf{read}(T')@v'.\langle\!\langle P\rangle\!\rangle_{u;V} \end{aligned}$$

$(*)$ $v \in V - \{\mathtt{self}\}$ and y, z are fresh

$(**)$ $\{x_1, \cdots, x_n\} = (fv(t, v) - \{\mathtt{self}\}) \cap V$ and $y, y_1, \cdots, y_n$ are fresh and
$t' = t[u, y_1, \cdots, y_n/\mathtt{self}, x_1, \cdots, x_n]$ and $v' = v[u, y_1, \cdots, y_n/\mathtt{self}, x_1, \cdots, x_n]$

$(***)$ $\{x_1, \cdots, x_n\} = (fv(T, v) - \{\mathtt{self}\}) \cap V$ and $y, y_1, \cdots, y_n$ are fresh and
$T' = T[u, y_1, \cdots, y_n/\mathtt{self}, x_1, \cdots, x_n]$ and $v' = v[u, y_1, \cdots, y_n/\mathtt{self}, x_1, \cdots, x_n]$

Table 9
Encoding KLAIM into μKLAIM: Components

The main encoding relies on an auxiliary encoding for node components given in Table 9. Then, the component C located in l is encoded as $(\!|C|\!)_{l;fv(C)}$. This encoding uses `env` for operations related to environments, keeps track of the locality where the component is located (to statically resolve occurrences of variable `self` and to dynamically enable the encoded term to properly translate the free variables occurring in actions **out**/**in**/**read**) and records the originally free variables occurring in C. This last information is necessary because the encoding proceeds compositionally; thus, it is necessary to distinguish which variables were free 'at the beginning' from those that are temporarily free but will be bound by a binding prefix during the encoding phase. To clarify this point, consider the following process

$$P \triangleq \mathbf{in}(!x_1)@l.\mathbf{out}(x_1, x_2)@l$$

located at l'. In this process, only x_2 is (originally) free. But to encode P, we need to first encode the (sub)process $\mathbf{out}(x_1, x_2)@l$ that has two free variables: x_1 and x_2. Hence, if we encode such a process as

$$\mathbf{read}(l', !y)@\mathtt{env}.\mathbf{read}(y, x_1, !y_1)@\mathtt{env}.\mathbf{read}(y, x_2, !y_2)@\mathtt{env}.\mathbf{out}(y_1, y_2)@l$$

we would change the overall behaviour. Indeed, the binding of the first argument of action **out** to the argument of action **in** (programmed in P) would be lost. The right solution is

$$\mathbf{read}(l', !y)@\mathtt{env}.\mathbf{read}(y, x_2, !y_2)@\mathtt{env}.\mathbf{out}(x_1, y_2)@l$$

that, once prefixed by (the encoding of) action $\mathbf{in}(!x_1)@l$, properly binds variable x_1.

To prove properties of this encoding, we first introduce a notion of *normal form* of an encoding $(\!|N|\!)$, written $(\!|\!|N|\!|\!)$. Essentially, the normal form of an encoding is the net resulting from the execution of (what we can call) *administrative* τ-steps. Informally, these are the τ-steps introduced by the encoding and that do not correspond to any τ-step in the source net. Normal forms enjoy the desirable property of being *prompt*, i.e. any top-level action they intend to perform corresponds to an analogous action in the source term. This fact will greatly simplify our proofs.

Intuitively, $(\!|\!|N|\!|\!)$ is obtained from $(\!|N|\!)$ by firing as many top-level 'administrative' actions **read** (introduced to implement allocation environments) as possible. For example, if ρ is the allocation environment of l and the side condition $(***)$ of Table 9 holds, we let

$$(\!|\!|\mathbf{read}(T)@v.P|\!|\!)_{l;V} \triangleq \mathbf{read}(l', x_k, !y_k)@\mathtt{env}.\cdots.\mathbf{read}(l', x_n, !y_n)@\mathtt{env}.$$
$$\mathbf{read}(T')@v'.(\!|P|\!)_{l;V}$$

where l' is the alias for l, $\{x_1, \cdots, x_{k-1}\} \subseteq dom(\rho)$ and $x_k \notin dom(\rho)$. The idea underlying this normalization is that, if $(\!|\!|\mathbf{read}(T)@v.P|\!|\!)_{l;V}$ has a top-level action

of the form **read**$(\cdot, x, !y)@\mathtt{env}$, then there exists a variable in $(fv(T,v)-\{\mathtt{self}\})\cap V$ that cannot be resolved in ρ; thus, the original action **read**$(T)@v$ gets stuck. Hence, as expected, also its encoding gets stuck when it tries to resolve variable x.

The above definition can be made more formal; however, for the sake of simplicity, we think that this intuitive presentation suffices. Just notice that the normalization procedure behaves similarly when the translated action is a **in**/**out**/**eval**, and it extends homomorphically to complex processes and nets. The following result states that the reduction to normal forms is performed while respecting $\gtrsim^{\mathrm{tr}}_{\mu K}$.

Lemma 3.1 $(\!|N|\!) \gtrsim^{\mathrm{tr}}_{\mu K} (\!|(\!|N|\!)|\!)$.

Proof: To prove the thesis, we need to show that

$$\mathfrak{R} \triangleq \{\ (\mathcal{C}[H], \mathcal{C}[K]) \ : (\!|N|\!) \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* H \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* K \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* (\!|(\!|N|\!)|\!) \wedge \ \mathcal{C}[\cdot] \text{ is a context translated via } (\!|\cdot|\!)\ \}$$

is contained in $\gtrsim^{\mathrm{tr}}_{\mu K}$. Let us pick up a pair $(\mathcal{C}[H], \mathcal{C}[K]) \in \mathfrak{R}$ and prove that it satisfies the requirements of the definition of $\gtrsim^{\mathrm{tr}}_{\mu K}$.

Let $\mathcal{C}[H] \xrightarrow{\chi} \overline{H}$ and let us reason by case analysis on χ.

- $\chi =$ **nil** $@\, l$**.** In this case, $\overline{H} \equiv \mathcal{C}[H]$; moreover, since H and K have the same addresses, it trivially holds that $\mathcal{C}[K] \xrightarrow{\chi} \mathcal{C}[K]$ and the thesis follows up-to $\equiv$.
- $\chi = (\nu \widetilde{l})\,\langle t\rangle\, @\, l$**.** If the datum is provided by the context, then the thesis is easy to prove. Otherwise, suppose that $H \xrightarrow{(\nu \widetilde{l'})\ \langle t\rangle\ @\ l} H'$ and let $\widetilde{l}$ be obtained from $\widetilde{l'}$ by adding some names $\widetilde{l''}$ bound by $\mathcal{C}[\cdot]$. Then, by definition of the encoding and of relation $\mathfrak{R}$, it must be that $N \xrightarrow{(\nu \widetilde{l'})\ \langle t\rangle\ @\ l} N'$ and that $(\!|N'|\!) \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* H' \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* K' \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* (\!|(\!|N'|\!)|\!)$, where $K \xrightarrow{(\nu \widetilde{l'})\ \langle t\rangle\ @\ l} K'$. In conclusion, $\mathcal{C}[H] \equiv (\nu \widetilde{l''})\mathcal{C}_1[H] \xrightarrow{\chi} \mathcal{C}_1[H']$, where $\mathcal{C}_1[\cdot]$ is still a translated context; moreover, $\mathcal{C}[K] \xrightarrow{\chi} \mathcal{C}_1[K']$ and $\mathcal{C}_1[H']\ \mathfrak{R}\ \mathcal{C}_1[K']$, as required.
- $\chi = \tau$**.** According to Lemma 2.9, we have six possible sub-cases, that we now examine separately.
  - (1) $H \xrightarrow{\tau} H'$ and $\overline{H} \equiv \mathcal{C}[H']$. There are two possibilities for this τ-step: it can be either generated by an action **read** over $\mathtt{env}$ or not.
    - (a) In the first case, by construction, it can be that K has been obtained from H by firing also such an action **read**; hence, $\mathcal{C}[K] \xrightarrow{\epsilon} \mathcal{C}[K]$ and $\mathcal{C}[H']\ \mathfrak{R}\ \mathcal{C}[K]$. Otherwise, K can mimic this τ-step and reduce to a K' such that $(\!|N|\!) \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* H' \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* K' \left(\xrightarrow{\cdot \triangleleft \mathtt{env}}\right)^* (\!|(\!|N|\!)|\!)$ and the thesis follows.
    - (b) On the other hand, if the τ-step of H did not involved any exchange over $\mathtt{env}$, it must be that K can perform the same action. Indeed, actions not involving $\mathtt{env}$ can only increase while passing from H to

K (no action over a locality different from env is touched and some new action over a locality different from env could be enabled by the removal of some prefixing **read** over env). Thus, $K \xrightarrow{\tau} K'$ such that $H' \left(\xrightarrow{\cdot \triangleleft \mathsf{env}}\right)^* K'$. We can conclude, once we prove that there exists a KLAIM net M such that $\langle\!\langle M \rangle\!\rangle \left(\xrightarrow{\cdot \triangleleft \mathsf{env}}\right)^* H'$ and $K' \left(\xrightarrow{\cdot \triangleleft \mathsf{env}}\right)^* \langle\!\langle\langle\!\langle M \rangle\!\rangle\!\rangle$. But this is not difficult: if H performs a τ-step without involving env, this means that N (that exists by definition of $\mathfrak{R}$) can perform a top-level τ-step over l, see the definition of the encoding in Tables 8 and 9. Then, the M we were looking for is the τ-reduct of N obtained from firing the action whose encoding has been fired by H.

(2) $\mathcal{C}[\cdot] \xrightarrow{\tau} \mathcal{C}'[\cdot]$ and $\overline{H} \equiv \mathcal{C}'[H]$. This case is trivial.

(3) $H \xrightarrow{\triangleright l} H'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}[\,\cdot \parallel l :: \mathbf{nil}]$ and $\overline{H} \equiv \mathcal{C}[H']$. Clearly, $l \neq \mathsf{env}$, otherwise H could have performed the τ-step without the contribution of the context. By definition of the normalization and of the relation $\mathfrak{R}$, K has as many sending actions as H (possibly, it has some more sending action resulting from the removal of some prefix **read**); thus, $K \xrightarrow{\triangleright l} K'$ such that $H' \left(\xrightarrow{\cdot \triangleleft \mathsf{env}}\right)^* K'$ and hence $\mathcal{C}[K] \xrightarrow{\tau} \mathcal{C}[K']$. Like in case 1.(b) above, we can find a net M such that $\langle\!\langle M \rangle\!\rangle \left(\xrightarrow{\cdot \triangleleft \mathsf{env}}\right)^* H'$ and $K' \left(\xrightarrow{\cdot \triangleleft \mathsf{env}}\right)^* \langle\!\langle\langle\!\langle M \rangle\!\rangle\!\rangle$: indeed, it is the $\triangleright l$ -reduct of N obtained from firing the action whose encoding has been fired by H.

(4) $H \xrightarrow{\mathbf{nil}\ @\ l} H'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\,\cdot \parallel L]$, $L \xrightarrow{\triangleright l} L'$ and $\overline{H} \equiv \mathcal{C}'[H \parallel L']$. This case is simpler.

(5) $H \xrightarrow{t \triangleleft l} H'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\,\cdot \parallel l :: \langle t \rangle]$ and $\overline{H} \equiv \mathcal{C}'[H']$. Again, $l \neq \mathsf{env}$, otherwise H could have performed the τ-step without the contribution of the context. The proof is like in case 3. above.

(6) $H \xrightarrow{(\nu \widetilde{l})\ \langle t \rangle\ @\ l} H'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\,\cdot \parallel L]$, $L \xrightarrow{t \triangleleft l} L'$ and $\overline{H} \equiv \mathcal{C}'[(\nu \widetilde{l})(H' \parallel L')]$. Like before.

The converse, i.e. that each χ-move of $\mathcal{C}[K]$ can be properly replied to by $\mathcal{C}[H]$, can be proved similarly. To prove closeness under translated contexts, let $\mathcal{D}[\cdot]$ be a translated context; we have to prove that $\mathcal{D}[\mathcal{C}[H]] \;\mathfrak{R}\; \mathcal{D}[\mathcal{C}[K]]$, but this holds by definition of $\mathfrak{R}$, once we consider the context $\mathcal{D}[\mathcal{C}[\cdot]]$ that is still a translated context. □

Now, we can consider the operational correspondence. Throughout this proof, we shall write ENV_l^ρ to indicate the tuples allocated at env to implement the allocation environment ρ of node l, i.e. $\langle l, l \rangle \mid \prod_{(x,l') \in \rho}^{x \neq \mathtt{self}} \langle l, x, l' \rangle$. To better understand the following proofs, notice that translated contexts comply with the expected interaction protocol with env. In particular, they cannot count how many times a given datum appears in env and cannot tell $\mathsf{env} :: ENV_l^\rho \mid \langle l', l \rangle$ and $\mathsf{env} :: ENV_l^\rho \mid ENV_{l'}^\rho$ apart.

Lemma 3.2 (Operational Correspondence) *Let N be a* KLAIM *net. Then*

(1) $N \longmapsto N'$ *implies that* $(\!|N|\!) \longmapsto^* \gtrsim^{\mathrm{tr}}_{\mu K} (\!|N'|\!)$
(2) $(\!|N|\!) \longmapsto N'$ *implies that* $N \longmapsto N''$ *and* $N' \gtrsim^{\mathrm{tr}}_{\mu K} (\!|N''|\!)$

Proof:

(1) The proof is by induction on the length of the inference for $N \longmapsto N'$. For the base case, we just consider two representative cases, i.e. when N evolves by exploiting rules (RED-IN) and (RED-NEW); the other ones are similar or easier.

In the first case, we have that $N \triangleq l ::_\rho \mathbf{in}(T)@u.P \parallel l' ::_{\rho'} \langle t \rangle$, and we let $V = fv(\mathbf{in}(T)@u.P)$. By hypothesis, $\rho(u) = l'$ and $\mathcal{E}[\![\ T\ ]\!]_\rho$ is defined and yields T'; thus, $fv(T,u) \subseteq dom(\rho)$. By construction, we have that

$$(\!|N|\!) \triangleq l :: \mathbf{in}(T')@l'.(\!|P|\!)_{l;V} \parallel l' :: \langle t \rangle \parallel \mathsf{env} :: ENV^{\rho}_{l} \mid ENV^{\rho'}_{l'}$$

Moreover, we also know that $match(T', t) = \sigma$. By using Lemma 3.1, we can conclude that $(\!|N|\!) \longmapsto (\!|l ::_\rho P\sigma \parallel l' ::_{\rho'} \mathbf{nil}|\!) \triangleq (\!|N'|\!) \gtrsim^{\mathrm{tr}}_{\mu K} (\!|N'|\!)$, as required.

When N evolves exploiting rule (RED-NEW), then $N \triangleq l ::_\rho \mathbf{new}(l').P$ and $N' \triangleq (\nu l')(l ::_\rho P \parallel l' ::_{\rho[l'/\mathtt{self}]} \mathbf{nil})$. It is easy to show that

$$\begin{aligned}(\!|N|\!) &\longmapsto^* (\nu l')(l :: (\!|P|\!)_{l,fv(P)} \parallel l' :: \mathbf{nil} \parallel \mathsf{env} :: ENV^{\rho}_{l} \mid \langle l', l \rangle) \\ &\gtrsim^{\mathrm{tr}}_{\mu K} (\nu l')(l :: (\!|P|\!)_{l,fv(P)} \parallel l' :: \mathbf{nil} \parallel \mathsf{env} :: ENV^{\rho}_{l} \mid ENV^{\rho}_{l'}) \gtrsim^{\mathrm{tr}}_{\mu K} (\!|N'|\!)\end{aligned}$$

We now consider the inductive step; we only discuss the case in which the last rule applied is (RED-STRUCT). In this case, $N \longmapsto N'$ because $N \equiv M$, $M \longmapsto M'$ and $M' \equiv N'$. It is easy to see that structurally equivalent nets have encodings related by $\lesssim^{\mathrm{tr}}_{\mu K}$; thus, $(\!|N|\!) \gtrsim^{\mathrm{tr}}_{\mu K} (\!|M|\!)$ and $(\!|M'|\!) \gtrsim^{\mathrm{tr}}_{\mu K} (\!|N'|\!)$. By induction, we know that $(\!|M|\!) \longmapsto^* M'' \gtrsim^{\mathrm{tr}}_{\mu K} (\!|M'|\!)$, for some M''. These two facts together imply that $(\!|N|\!) \longmapsto^* \overline{N}$ for some $\overline{N}$ such that $\overline{N} \gtrsim^{\mathrm{tr}}_{\mu K} M''$. By transitivity of $\gtrsim^{\mathrm{tr}}_{\mu K}$, we can conclude.

(2) The proof is by induction on the length of the inference for $(\!|N|\!) \longmapsto N'$. We only examine the base cases for (RED-IN) and (RED-NEW). The key observation is that, because of normalization, $(\!|N|\!)$ can evolve via rule (RED-IN) only if

$$(\!|N|\!) \triangleq l :: \mathbf{in}(T')@l'.(\!|P|\!)_{l;V} \parallel l' :: \langle t \rangle \parallel \mathsf{env} :: ENV^{\rho}_{l} \mid ENV^{\rho'}_{l'}$$

where $V = fv(\mathbf{in}(T')@l'.P)$ and $match(T', t) = \sigma$; moreover, we also have that

$$N' \equiv l :: (\!|P\sigma|\!)_{l;V} \parallel l' :: \mathbf{nil} \parallel \mathsf{env} :: ENV^{\rho}_{l} \mid ENV^{\rho'}_{l'}$$

Now, it must be that $N \triangleq l ::_\rho \mathbf{in}(T)@u.P \parallel l' ::_{\rho'} \langle t \rangle$, where $\rho(u) = l'$ and $\mathcal{E}[\![\ T\ ]\!]_\rho = T'$. This suffices to infer $N \longmapsto l ::_\rho P\sigma \parallel l' ::_{\rho'} \mathbf{nil} \triangleq N''$ and $N' \equiv (\!|N''|\!)$.

The case for (RED-NEW) is proved like before. Indeed, $N \triangleq l ::_\rho \mathbf{new}(l').P$, $(\!|\!|N|\!|\!|) \triangleq (\!|N|\!)$ and $N' \equiv l :: \mathbf{read}(l, !y)@\mathsf{env}.\mathbf{out}(l', y)@\mathsf{env}.(\!|P|\!)_{l,fv(P)} \parallel l' :: \mathbf{nil} \parallel \mathsf{env} :: ENV_l^\rho \mid \langle l', l\rangle$. Thus, $N \longmapsto (\nu l')(l ::_\rho P \parallel l' ::_{\rho[l'/\mathtt{self}]} \mathbf{nil}) \triangleq N''$ and

$$\begin{aligned} N' &\gtrsim^{\mathrm{tr}}_{\mu K} (\nu, l')(l :: (\!|P|\!)_{l,fv(P)} \parallel l' :: \mathbf{nil} \parallel \mathsf{env} :: ENV_l^\rho \mid \langle l', l\rangle) \\ &\gtrsim^{\mathrm{tr}}_{\mu K} (\nu, l')(l :: (\!|P|\!)_{l,fv(P)} \parallel l' :: \mathbf{nil} \parallel \mathsf{env} :: ENV_l^\rho \mid ENV_{l'}^{\rho'}) \triangleq (\!|\!|N''|\!|\!|) \end{aligned}$$

that can be easily proved. □

Theorem 3.3 (Full Abstraction w.r.t. Translated Barbed Congruence) *$N \cong_K M$ if and only if $(\!|N|\!) \cong^{\mathtt{tr}}_{\mu K} (\!|M|\!)$.*

Proof: We start with the 'if' part and prove that $\mathfrak{R} \triangleq \{(N, M) : (\!|\!|N|\!|\!|) \cong^{\mathtt{tr}}_{\mu K} (\!|\!|M|\!|\!|)\}$ is barb preserving, reduction closed and contextual. Indeed, by Lemma 3.1, $(\!|\cdot|\!) \gtrsim^{\mathrm{tr}}_{\mu K} (\!|\!|\cdot|\!|\!|)$; hence, the hypothesis $(\!|N|\!) \cong^{\mathtt{tr}}_{\mu K} (\!|M|\!)$ implies that $(\!|\!|N|\!|\!|) \cong^{\mathtt{tr}}_{\mu K} (\!|\!|M|\!|\!|)$, as needed.

- Let $N \downarrow l$; since the encoding and the normalization preserve the barbs (this can be easily seen by the definitions of $(\!|\cdot|\!)$ and $(\!|\!|\cdot|\!|\!|)$), we have that $(\!|\!|N|\!|\!|) \downarrow l$. Then, by hypothesis, $(\!|\!|M|\!|\!|) \Downarrow l$, i.e. $(\!|\!|M|\!|\!|) \longmapsto^* M' \downarrow l$. Now, by Lemmata 3.2.2 and 3.1, we have that there exists a net M'' such that $M \longmapsto^* M''$ and $M' \gtrsim^{\mathrm{tr}}_{\mu K} (\!|\!|M''|\!|\!|)$. By Definition 2.3 and Proposition 2.8.2, we can conclude that $M'' \downarrow l$ and hence $M \Downarrow l$.
- Let $N \longmapsto N'$; by Lemma 3.2.1 this implies that $(\!|\!|N|\!|\!|) \longmapsto^* \gtrsim^{\mathrm{tr}}_{\mu K} (\!|\!|N'|\!|\!|)$. By hypothesis, we have that $(\!|\!|M|\!|\!|) \longmapsto^* \gtrsim^{\mathrm{tr}}_{\mu K} M'$, for some M' such that $(\!|\!|N'|\!|\!|) \cong^{\mathtt{tr}}_{\mu K} M'$. By Lemmata 3.2.2 and 3.1, we have that there exists a net M'' such that $M \longmapsto^* M''$ and $M' \gtrsim^{\mathrm{tr}}_{\mu K} (\!|\!|M''|\!|\!|)$. Now, since $\lesssim^{\mathrm{tr}}_{\mu K} \subseteq \cong^{\mathtt{tr}}_{\mu K}$ (that can be easily verified) and by transitivity of $\cong^{\mathtt{tr}}_{\mu K}$, we have that $(\!|\!|N'|\!|\!|) \cong^{\mathtt{tr}}_{\mu K} (\!|\!|M''|\!|\!|)$; thus, $N' \mathfrak{R} M''$, as required.
- Let us pick up a translated context $\mathcal{C}[\cdot]$; this means that $\mathcal{C}[\cdot] \triangleq (\!|\mathcal{D}[\cdot]|\!)$. Now, if either $\mathcal{D}[N]$ or $\mathcal{D}[M]$ are undefined (i.e. they give rise to a ill-defined net) then we do not have to consider $\mathcal{D}[\cdot]$ for context closure of $\mathfrak{R}$. Otherwise, we have to prove that $\mathcal{D}[N] \mathfrak{R} \mathcal{D}[M]$ by knowing that $\mathcal{C}[(\!|\!|N|\!|\!|)] \cong^{\mathtt{tr}}_{\mu K} \mathcal{C}[(\!|\!|M|\!|\!|)]$. By Lemma 3.1 (that can be easily extended to contexts) we have that $\mathcal{C}[\cdot] \gtrsim^{\mathrm{tr}}_{\mu K} (\!|\!|\mathcal{D}[\cdot]|\!|\!|)$ and hence $\mathcal{C}[\cdot] \gtrsim^{\mathrm{tr}}_{\mu K} (\!|\!|\mathcal{D}|\!|\!|)[\cdot]$; thus, $(\!|\!|\mathcal{D}|\!|\!|)[(\!|\!|N|\!|\!|)] \cong^{\mathtt{tr}}_{\mu K} (\!|\!|\mathcal{D}|\!|\!|)[(\!|\!|M|\!|\!|)]$, i.e. $(\!|\!|\mathcal{D}[N]|\!|\!|) \cong^{\mathtt{tr}}_{\mu K} (\!|\!|\mathcal{D}[M]|\!|\!|)$. By definition, we obtain the required $\mathcal{D}[N] \mathfrak{R} \mathcal{D}[M]$.

Conversely, we can similarly prove that $\mathfrak{R} \triangleq \{((\!|\!|N|\!|\!|), (\!|\!|M|\!|\!|)) : N \cong_K M\}$ is barb preserving, reduction closed and contextual. We omit the details, since they are an easy adaption of the above steps. The only tricky part is barb preservation when $(\!|\!|N|\!|\!|) \downarrow \mathsf{env}$; however, since $(\!|\!|M|\!|\!|)$ always has at least one (possibly useless) datum at env, we also have that $(\!|\!|M|\!|\!|) \downarrow \mathsf{env}$, as required. □

To conclude this section, we want to stress that we need env not to be empty to preserve, e.g., the equivalence $\mathbf{0} \cong_K (\nu l)(l ::_{[\mathtt{self}\mapsto l]} \mathbf{nil})$. Once translated, these two nets become $\mathsf{env} :: \langle\rangle$ and $(\nu l)(l :: \mathbf{nil} \parallel \mathsf{env} :: \langle l, l\rangle)$ resp., that are equivalent w.r.t. $\cong^{\mathtt{tr}}_{\mu K}$ exactly because translated contexts cannot tell $\langle\rangle$ and $\langle l, l\rangle$ apart when located at env.

4 μKLAIM vs cKLAIM

In this section, we develop a fully abstract but possibly divergent encoding of μKLAIM in cKLAIM. As we already stressed, the main differences between these two dialects are due to presence/absence of action **read** and to polyadic/monadic communications. Action **read** is trivial to encode in the more elementary calculus. The encoding of polyadic communications into monadic ones is the most complex part of this work and provide evidence of the expressive power of LINDA's pattern matching mechanism.

We start with the easier task: proving that actions **read** can be implemented in cKLAIM.

Essentially, **read** behaves like **in** except for the fact that it does not remove the accessed datum. It is easy to prove that

$$l :: \mathbf{read}(T)@l'.P \;\cong_{\mu K}\; l :: \mathbf{in}(T)@l'.\mathbf{out}(\widehat{T})@l'.P$$

where $\widehat{l} \triangleq l$, $\widehat{!x} \triangleq x$ and $\widehat{T_1, T_2} \triangleq \widehat{T_1}, \widehat{T_2}$. This implementation of action **read** can be extended to complex nets in the obvious way, i.e. structurally; it can be then easily proved that the resulting encoding enjoys semantical equivalence w.r.t. $\cong_{\mu K}$. We omit the details on this aspect to leave space to the second difference between μKLAIM and cKLAIM, namely the use of polyadic/monadic data.

To softly introduce the reader to our encoding, first let us examine Milner's well-known encoding of polyadic into monadic communication for *synchronous* π-calculus [22]. We have that:

$$\bar{a}\langle b, c\rangle \quad | \quad a(x, y)$$

becomes

$$(\nu n)\,\bar{a}\langle n\rangle.\,\bar{n}\langle b\rangle.\,\bar{n}\langle c\rangle \quad | \quad a(n).n(x).n(y)$$

with n fresh. Hence, a fresh name (n) is exchanged by exploiting a common channel (a); n is then used to pass the sequence of values. In *asynchronous* π-calculus [21], Honda and Tokoro propose a slightly more complex encoding:

$$\bar{a}\langle b, c\rangle \quad | \quad a(x, y)$$

is rendered as

$$(\nu n)(\overline{a}\langle n\rangle \mid n(n_1).(\overline{n_1}\langle b\rangle \mid n(n_2).\overline{n_2}\langle c\rangle))$$
$$\mid \quad a(n).(\nu n_1, n_2)(\overline{n}\langle n_1\rangle \mid n_1(x).(\overline{n}\langle n_2\rangle \mid n_2(y)))$$

The schema is similar to the one for the synchronous calculus. However, since output sequenzialization is not possible, different channels are needed to send the different values in the sequence.

Our encoding somehow evolves Honda's one because it also has to consider the presence of pattern-matching. Hence, when encoding a polyadic communication (of μKLAIM) into a monadic one (of cKLAIM) we are faced with the problem of starting to access a tuple and, while scanning it, finding out that it does not match the specified template. The solution is to then put back the part of the tuple retrieved and restart the process; of course, this introduces divergence in the encoding. The full encoding is given in Table 10.

Remark 4.1 The encoding in Table 10 is defined only for nets in which each tuple is located alone on a different clone of the node hosting it (thus, for example, $\langle\!| l :: \langle t_1\rangle | \langle t_2\rangle |\!\rangle$ is not defined). To overcome this problem and let the encoding easy, we let $\langle\!| N |\!\rangle$ to be $\langle\!| N' |\!\rangle$ where $N' \equiv N$ but $\langle\!| N' |\!\rangle$ is defined. Notice that such N' can be always found for each N by only using rule (CLONE), but is not unique, in general (indeed, we can also split processes located at the same locality). To overcome this fact, we can consider the N' (unique up-to rearrangements of parallel components) obtained from N by only using (CLONE) to isolate located data.

The focus of the encoding is in the implementation of tuples and in the translation of actions **in**/**out**. A tuple $\langle t\rangle$ is translated into a (monadic) reference to a fresh locality l where a process, $R_l(t)$, sequentially produces the fields of the tuple and the length of the tuple plus one (this is used to properly implement the pattern matching mechanism). The fields are requested sequentially by the (translation of an) action **in** by using localities $1, 2, \ldots, n, \ldots$; this is necessary to maintain the order of the data in the tuple, since our calculus is asynchronous. Once the process $R_l(t)$ has accepted the requirement for the i-th field, it produces such a field together with an acknowledgement implemented via the reserved locality go.

Once the process translating an action **in** acquires the reference to (the locality hosting the process handling) a tuple, it first verifies whether the accessed tuple and the template used to retrieve it have the same number of fields. If this is the case, it sequentially asks for all the fields of the tuple. For the i-th tuple field u_i, the encoding of the input non-deterministically chooses whether accepting u_i (because it matches the i-th template parameter T_i), thus proceeding with the tuple scanning, or refusing it and re-establishing the original situation (with the reference put back in its original location and the process handling the tuple rolled back). In the latter case, notice that the input has not been fired and hence the process implementing

Encoding Nets:

$$\langle\!|\mathbf{0}|\!\rangle \triangleq \mathbf{0} \qquad \langle\!|N_1 \parallel N_2|\!\rangle \triangleq \langle\!|N_1|\!\rangle \parallel \langle\!|N_2|\!\rangle \qquad \langle\!|(\nu l)N|\!\rangle \triangleq (\nu l)\langle\!|N|\!\rangle$$

$$\langle\!|l :: C|\!\rangle \triangleq \begin{cases} (\nu l')(l :: \langle l' \rangle \parallel l' :: R_{l'}(t)) & \text{if } C \triangleq \langle t \rangle \text{ and } l' \text{ is fresh} \\ l :: \langle\!|P|\!\rangle & \text{if } C = P \end{cases}$$

Encoding Processes:

$$\langle\!|\mathbf{nil}|\!\rangle \triangleq \mathbf{nil} \qquad \langle\!|X|\!\rangle \triangleq X \qquad \langle\!|\mathbf{rec}\, X.P|\!\rangle \triangleq \mathbf{rec}\, X.\langle\!|P|\!\rangle \qquad \langle\!|P_1|P_2|\!\rangle \triangleq \langle\!|P_1|\!\rangle \mid \langle\!|P_2|\!\rangle$$

$$\langle\!|\mathbf{new}(l).P|\!\rangle \triangleq \mathbf{new}(l).\langle\!|P|\!\rangle \qquad \langle\!|\mathbf{eval}(Q)@l.P|\!\rangle \triangleq \mathbf{eval}(\langle\!|Q|\!\rangle)@l.\langle\!|P|\!\rangle$$

$$\langle\!|\mathbf{out}(t)@l.P|\!\rangle \triangleq \mathbf{eval}(\mathbf{nil})@l.\mathbf{new}(l').\mathbf{out}(l')@l.\mathbf{eval}(R_{l'}(t))@l'.\langle\!|P|\!\rangle \quad \text{with } l' \text{ fresh}$$

$$\langle\!|\mathbf{in}(T)@l.P|\!\rangle \triangleq \mathbf{rec}\, X.\mathbf{in}(!x)@l.Q^0_{l,x,X}(T;P) \qquad \text{with } x, X \text{ fresh}$$

---

where:

- $R_l(u_1,\ldots,u_n) \triangleq S_l(u_1,\ldots,u_n) \mid L^n_l$
- $S_l(u_1,\ldots,u_n) \triangleq \prod_{i=1}^{n} \mathbf{in}(i)@l.\mathbf{new}(l_i).\mathbf{out}(\mathtt{go})@l.\mathbf{out}(l_i)@l.\mathbf{out}(u_i)@l_i$ with l_i fresh
- $L^n_l \triangleq \mathbf{in}(\mathtt{len})@l.\mathbf{new}(l_{len}).\mathbf{out}(\mathtt{go})@l.\mathbf{out}(l_{len})@l.\mathbf{out}(n+1)@l_{len}$ with l_{len} fresh
- $$Q^k_{l,x,X}(T_1,\ldots,T_n;P) \triangleq \begin{cases} \mathbf{out}(\mathtt{len})@x.\mathbf{in}(\mathtt{go})@x.\mathbf{in}(!x_{len})@x.(\\ \quad \mathbf{in}(n+1)@x_{len}.Q^1_{l,x,X}(T_1,\ldots,T_n;P) \\ \quad \mid \mathbf{in}(!y)@x_{len}.\mathbf{eval}(L^n_l)@x.\mathbf{out}(x)@l.X\ ) & \text{if } k = 0 \\ \mathbf{out}(k)@x.\mathbf{in}(\mathtt{go})@x.\mathbf{in}(!x_k)@x.(\\ \quad \mathbf{in}(T_k)@x_k.Q^{k+1}_{l,x,X}(T_1,\ldots,T_n;P) \\ \quad \mid \mathbf{in}(!y)@x_k.\mathbf{eval}\big(\, L^n_x \mid S_x(\widehat{T_1},\ldots,\widehat{T_k})\,\big)@x.\mathbf{out}(x)@l.X\ ) & \text{if } 1 \le k \le n \\ \langle\!|P|\!\rangle & \text{if } k = n+1 \end{cases}$$

  with x_{len}, y and x_k fresh variables
- $\widehat{T} \triangleq \begin{cases} u & \text{if } T = u \\ x & \text{if } T = !x \end{cases}$
- $\mathtt{len}, \mathtt{go}, 1, \ldots, n, \ldots$ are pairwise distinct reserved localities

Table 10
Encoding The Polyadic Calculus into the Monadic Calculus

Tuple Consumer (with template T)		Tuple Handler (for tuple t)
Acquire the **lock** over a tuple		
Ask for t's **length**	$\longrightarrow$	
	$\longleftarrow$	Provide t's length k
If $k = \|T\|$ then proceed, otherwise release the lock and roll back the tuple handler		
Ask for t's **first field**	$\longrightarrow$	
	$\longleftarrow$	Provide t's first field f_1 and an ack go
If the first field of T matches f_1 then proceed, otherwise release the lock and roll back the tuple handler		
...	...	...
Ask for t's **last field**	$\longrightarrow$	
	$\longleftarrow$	Provide t's the last field f_k and an ack go
If the last field of T matches f_k then FINISH, otherwise release the lock and roll back the tuple handler		

Table 11
The Protocol to Encode Polyadic Communications

it recursively starts back its task. Clearly, this protocol is *not divergent free*; the intuition underlying it is illustrated in Table 11.

We now prove some interesting properties of $\langle\!|\cdot|\!\rangle$. In particular, we prove that a polyadic net N and its encoding are semantically equivalent w.r.t. barbed bisimulation (clearly, they cannot be equivalent w.r.t. any equivalence that is a congruence). We also prove that the encoding is *adequate w.r.t. barbed congruence*, but it is *not* fully abstract (at least, when considering all the possible monadic contexts in the context closure). Like for π-calculus [3], a fully abstract encoding seems very hard to achieve. The problem is that putting two encoded terms in a generic context (i.e., a context not necessarily corresponding to the encoding of any term) can break the equivalence. In our setting, consider the polyadic net $N \triangleq l :: \mathbf{in}(!x)@l.\mathbf{out}(x)@l$; it easy is to prove that $N \cong_{\mu K} l :: \mathbf{nil}$. However, $\langle\!|N|\!\rangle \cong_{cK} \langle\!|l :: \mathbf{nil}|\!\rangle$ does not hold:

[3] [31] shows that Milner's encoding (sketched before) is *not* fully abstract w.r.t. bisimulation.

e.g., contestuality is broken by the context $[\_] \parallel l :: \langle l \rangle | \langle 2 \rangle$ that provides a link to an 'unfair' tuple handler (actually, it provides a non-restricted locality and the handler only provides the length of a tuple but not its fields). Indeed, the protocol of Table 11 cannot succeed because N gets blocked in $Q^1_{\ldots}$ since no go will be ever produced at l.

We believe that, by relying on sophisticated typing theories (like in [31]) to consider in the context closure only those contexts that do not violate the exchange protocol implemented by the encoding, a (restricted) fully abstraction result can be proved. However, it seems us unreasonable for a tuple-based language to assume that the repository of a node contains only data (i.e. tuples) of the same kind (i.e. with the same shape). So, even if theoretically possible, fully abstraction (w.r.t. an equivalence that is a congruence) would be in contrast with the principles underlying the tuple-space paradigm.

We now give the theoretical results. They rely on some preliminary steps, describing the operational correspondence between polyadic nets and their encoded monadic nets.

Proposition 4.2 *The following facts hold.*

(1) If $N \xrightarrow{\mathbf{nil}\ @\ l} N'$ *then* $(\!|N|\!) \xrightarrow{\mathbf{nil}\ @\ l} (\!|N'|\!)$. *Viceversa, if* $(\!|N|\!) \xrightarrow{\mathbf{nil}\ @\ l} M$ *then* $N \xrightarrow{\mathbf{nil}\ @\ l} N'$ *and* $M \triangleq (\!|N'|\!)$.

(2) If $N \xrightarrow{(\nu \widetilde{l})\ \langle t \rangle\ @\ l} N'$ *then* $(\!|N|\!) \xrightarrow{(\nu l')\ \langle l' \rangle\ @\ l} (\nu \widetilde{l})((\!|N'|\!) \parallel l :: \mathbf{nil} \parallel l' :: R_{l'}(t))$. *Viceversa, if* $(\!|N|\!) \xrightarrow{(\nu l')\ \langle l' \rangle\ @\ l} M$ *then* $N \xrightarrow{(\nu \widetilde{l})\ \langle t \rangle\ @\ l} N'$ *and* $M \equiv (\nu \widetilde{l})((\!|N'|\!) \parallel l :: \mathbf{nil} \parallel l' :: R_{l'}(t))$.

(3) If $N \xrightarrow{\triangleright\, l} N'$ *then* $(\!|N|\!) \stackrel{\triangleright\, l}{\Longrightarrow} (\!|N'|\!)$. *Viceversa, if* $(\!|N|\!) \xrightarrow{\triangleright\, l} M$, *then* $N \xrightarrow{\triangleright\, l} N'$ *and* $M \gtrsim_{cK} (\!|N'|\!)$.

(4) If $N \xrightarrow{t \triangleleft l} N'$ *then* $(\!|N|\!) \xrightarrow{l' \triangleleft l} (\!|\mathcal{C}|\!)\left[l'' :: Q^0_{l,l',X}(T;P)[\mathbf{in}(!x)@l.Q^0_{l,x,X}(T;P)/X]\right]$, *where* $N \equiv \mathcal{C}[l'' :: \mathbf{in}(T)@l.P]$ *for some* $\mathcal{C}[\cdot]$, l'', T *and* P *such that* $fn(l,t) \cap bn(\mathcal{C}[\cdot]) = \emptyset$ *and* $match(T,t) = \sigma$. *Viceversa, if* $(\!|N|\!) \xrightarrow{l' \triangleleft l} N_1$ *then* $N \equiv \mathcal{C}[l'' :: \mathbf{in}(T)@l.P]$ *for some* $\mathcal{C}[\cdot]$, l'', T, *and* P *such that* $l \notin bn(\mathcal{C}[\cdot])$. *Moreover, for every* t *s.t.* $fn(t) \cap bn(\mathcal{C}[\cdot]) = \emptyset$ *and* $match(T,t) = \sigma$, *it holds that* $N \xrightarrow{t \triangleleft l} \mathcal{C}[l'' :: P\sigma]$.

Proof: All the statements can be proved by induction on the inference length. The proof is long and standard, thus we omit it. □

Lemma 4.3 (Preservation of Execution Steps) *If* N *is a polyadic net and* $N \xrightarrow{\tau} N'$ *then* $(\!|N|\!) \Rightarrow \gtrsim_{cK} (\!|N'|\!)$.

Proof: The proof is by induction on the length of the inference for $\xrightarrow{\tau}$. There are three base cases: when using (LTS-New), (LTS-Send) and (LTS-Comm). The first

one is straightforward; we now inspect the other cases.

(LTS-Send**).** We have that $N \triangleq N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N'$ because $N_1 \xrightarrow{\triangleright\, l} N_1'$ and $N_2 \xrightarrow{\mathbf{nil}\ @\ l} N_2'$. In this case, we use Propositions 4.2.1 and .3 to conclude that $\langle\!|N|\!\rangle \Rightarrow \langle\!|N_1'|\!\rangle \parallel \langle\!|N_2'|\!\rangle \parallel l :: \langle\!|P|\!\rangle \equiv \langle\!|N'|\!\rangle$.

(LTS-Comm**).** We have that $N \triangleq N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N'$ because $N_1 \xrightarrow{\langle t\rangle\, \triangleleft\, l} N_1'$ and $N_2 \xrightarrow{\langle t\rangle\ @\ l} N_2'$. Then, by using Propositions 4.2.2 and .4 and Proposition 2.16.5, we can say that $\langle\!|N|\!\rangle \Rightarrow (\nu l')(\langle\!|N'|\!\rangle \parallel l' :: \mathbf{nil}) \gtrsim_{cK} \langle\!|N'|\!\rangle$ because $l' \notin n(N)$ and, thus, $l' \notin n(\langle\!|N'|\!\rangle)$.

For the inductive case, we analyse the last rule used, namely (LTS-Par), (LTS-Res) and (LTS-Struct). All the cases are easy. □

Let us now focus on the converse; to this aim, we need a slightly more involved result. We start with a definition needed to consider the intermediate states in the execution of a communication. Recall that $l \in fl(N)$ if and only if $N \equiv N' \parallel l :: \mathbf{nil}$.

Definition 4.4 *(1) A* cKlaim *net M is a* partial reduct *of a* μKlaim *net N whenever $N \equiv l_1 :: \mathbf{in}(T)@l_2.P \parallel l_2 :: \langle t\rangle$ and $\langle\!|N|\!\rangle \stackrel{\tau}{\Rightarrow} M \stackrel{\tau}{\Rightarrow} \gtrsim_{cK} \langle\!|N|\!\rangle$.*

(2) A cKlaim *net M is a* partial state *of a* μKlaim *net N whenever $N \equiv (\nu\widetilde{l})(N_1 \parallel \cdots \parallel N_n \parallel \overline{N})$, $M \equiv (\nu\widetilde{l})(M_1 \parallel \cdots \parallel M_n \parallel \langle\!|\overline{N}|\!\rangle)$ and for all i it holds that $fl(N_i) \subseteq fl(\overline{N})$ and that M_i is a partial reduct of N_i.*

A pleasant property of partial reducts (that turns out to be crucial for the proof of Theorem 4.8) now follows.

Lemma 4.5 *If M is a partial reduct of $N \equiv l_1 :: \mathbf{in}(T)@l_2.P \parallel l_2 :: \langle t\rangle$ and $M \xrightarrow{\tau} M'$, then either M' is a partial reduct of N, or $M' \gtrsim_{cK} \langle\!|N|\!\rangle$, or $M' \gtrsim_{cK} l_1 :: \langle\!|P\sigma|\!\rangle \parallel l_2 :: \mathbf{nil}$, where $\sigma = match(T, t)$.*

Proof: By definition of partial reduct and by an easy inspection of the possible reductions. □

Lemma 4.6 (Reflection of Execution Steps) *If N is a polyadic net and $\langle\!|N|\!\rangle \xrightarrow{\tau} M$ then either $N \xrightarrow{\tau} N'$ and $M \gtrsim_{cK} \langle\!|N'|\!\rangle$, or M is a partial state of N.*

Proof: The proof is by induction on the length of the inference for $\langle\!|N|\!\rangle \xrightarrow{\tau} M$ having τ as label. There are three base cases:

(LTS-Send**).** In this case it holds that $\langle\!|N|\!\rangle \triangleq \langle\!|N_1|\!\rangle \parallel \langle\!|N_2|\!\rangle \xrightarrow{\tau} M$. Then, by definition, $\langle\!|N_1|\!\rangle \xrightarrow{\triangleright\, l} M_1$, $\langle\!|N_2|\!\rangle \xrightarrow{\mathbf{nil}\ @\ l} M_2$ and $M \triangleq M_1 \parallel M_2$. By Proposition 4.2.3, we know that $N_1 \xrightarrow{\triangleright\, l} N_1'$ and $M_1 \gtrsim_{cK} \langle\!|N_1'|\!\rangle$. Thus, $N \xrightarrow{\tau} N_1' \parallel N_2 \triangleq N'$ and $M \gtrsim_{cK} \langle\!|N'|\!\rangle$.

(LTS-Comm**).** In this case it holds that $\langle\!|N|\!\rangle \triangleq \langle\!|N_1|\!\rangle \parallel \langle\!|N_2|\!\rangle \xrightarrow{\tau} M_1 \parallel M_2 \triangleq M$ because $\langle\!|N_1|\!\rangle \xrightarrow{l' \triangleleft l} M_1$ and $\langle\!|N_2|\!\rangle \xrightarrow{\langle l' \rangle @ l} M_2$. This case is *not* possible, since no encoding of a net can directly offer a non-restricted datum.

(LTS-New**).** This case trivially falls in the first possibility of this Lemma.

For the inductive case, we reason by case analysis on the last rule used in the inference.

(LTS-Par**).** In this case it holds that $\langle\!|N|\!\rangle \triangleq \langle\!|N_1|\!\rangle \parallel \langle\!|N_2|\!\rangle \xrightarrow{\tau} M' \parallel \langle\!|N_2|\!\rangle \triangleq M$ because $\langle\!|N_1|\!\rangle \xrightarrow{\tau} M'$. By induction, either $N_1 \xrightarrow{\tau} N_1'$ and $M' \gtrsim_{cK} \langle\!|N_1'|\!\rangle$, or M' is a partial state of N_1. In the first case, we have that $N \xrightarrow{\tau} N_1' \parallel N_2 \triangleq N'$ and $M \gtrsim_{cK} \langle\!|N'|\!\rangle$. In the second case, M is a partial state of N, by definition.

(LTS-Res**).** We now isolate two sub-cases:

- $\langle\!|N|\!\rangle \triangleq \langle\!|(\nu l)N_1|\!\rangle \triangleq (\nu l)\langle\!|N_1|\!\rangle \xrightarrow{\tau} (\nu l)M_1 \triangleq M$ because $\langle\!|N_1|\!\rangle \xrightarrow{\tau} M_1$. This case easily follows by induction.
- $\langle\!|N|\!\rangle \triangleq (\nu l)M_1 \xrightarrow{\tau} (\nu l)M_2 \triangleq M$ because $M_1 \xrightarrow{\tau} M_2$ but M_1 is not the encoding of any polyadic net. In this case, locality l is fresh for N and has been introduced by the encoding. It is then easy to see that l is the reference for a datum located in a node of N, i.e. $M_1 \triangleq l_1 :: \langle l \rangle \parallel l :: R_l(t)$, and $N \triangleq l_1 :: \langle t \rangle$. But then no τ-step can be performed by M_1

(LTS-Struct**).** In this case we have that $\langle\!|N|\!\rangle \equiv M_1 \xrightarrow{\tau} M_2 \equiv M$. Let $A_N \triangleq bn(\langle\!|N|\!\rangle) - n(N)$ be the (restricted) names introduced by the encoding; we then proceed by induction on k, the number of names in A_N touched by rules (RCom) and (Ext) in deriving $\langle\!|N|\!\rangle \equiv M_1$. We shall refer to this latter induction as the *internal* induction, while the induction on the number of rules used to infer $\xrightarrow{\tau}$ will be called the *external* one.

Base. If $k = 0$, then we can claim that $M_1 \triangleq \langle\!|N''|\!\rangle$ for some $N'' \equiv N$ (this can be proved by an easy induction on the length of $\langle\!|N|\!\rangle \equiv M_1$). By using this fact and a straightforward external induction, the thesis holds easily.

Induction. Let $l \in A_N$. Then $\langle\!|N|\!\rangle \triangleq \mathcal{C}[(\nu l)(l' :: \langle l \rangle \parallel l :: R_l(t))]$ for some t and l'. By using Lemma 2.9 and a simple analysis over the definition of the involved processes, it can only be one of the following cases:

- $\mathcal{C}[\mathbf{0}] \xrightarrow{\tau} \mathcal{C}'[\mathbf{0}]$ and $M \equiv \mathcal{C}'[(\nu l)(l' :: \langle l \rangle \parallel l :: R_l(t))]$. But then $\langle\!|N|\!\rangle \triangleq \mathcal{C}[(\nu l)(l' :: \langle l \rangle \parallel l :: R_l(t))] \xrightarrow{\tau} M$ can be inferred without touching l with rules (RCom) and (Ext) Hence, by internal induction, we can conclude.
- $\mathcal{C}[\cdot] \triangleq \mathcal{C}_1[\mathcal{C}_2[\cdot] \parallel H]$ where $H \xrightarrow{\triangleright l'} H'$, $l' \notin bn(\mathcal{C}_2[\cdot])$ and $M \equiv \mathcal{C}_1[(\widetilde{\nu l})(\mathcal{C}_2[l' :: \langle l \rangle \parallel l :: R_l(t)] \parallel H')]$. Like in the previous case, $\langle\!|N|\!\rangle \triangleq \mathcal{C}[(\nu l)(l' :: \langle l \rangle \parallel l :: R_l(t))] \xrightarrow{\tau} M$ can be inferred without touching l with rules (RCom) and (Ext) Hence, by internal induction, we can conclude.
- $\mathcal{C}[\cdot] \triangleq \mathcal{C}_1[\mathcal{C}_2[\cdot] \parallel H]$ where $H \xrightarrow{l \triangleleft l'} H'$ and $M \equiv \mathcal{C}_1[\mathcal{C}_2[(\nu l)(l' :: \mathbf{nil} \parallel l :: R_l(t)] \parallel H')]$. By Proposition 4.2.4, $H \triangleq \mathcal{E}[l'' :: \mathbf{rec}\, X.\mathbf{in}(!x)@l'.Q^0_{l',x,X}(T;P)]$, for some $\mathcal{E}[\cdot]$, l'', T and P, where con-

text $\mathcal{E}[\cdot]$ does not bind l and l'; moreover, $H' \equiv \mathcal{E}[l'' :: P']$, where $P' \triangleq Q^0_{l',l,X}(T;P)[\mathbf{rec}\, X.\mathbf{in}(!x)@l'.Q^0_{l',x,X}(T;P)/X]$. Thus,

$$M \equiv \mathcal{C}_1[\mathcal{C}_2[(\nu l)(l' :: \mathbf{nil} \parallel l :: R_l(t)] \parallel l'' :: P') \parallel l' :: \mathbf{nil} \parallel \mathcal{E}[l'' :: \mathbf{nil}]]$$

Now, we have that

$$N \equiv \mathcal{D}[(l' :: \langle t \rangle \parallel l'' :: \mathbf{in}(T)@l'.P) \parallel l' :: \mathbf{nil} \parallel \mathcal{F}[l'' :: \mathbf{nil}]]$$

for $\mathcal{E}[\cdot] \triangleq (\!|\mathcal{F}[\cdot]|\!)$ and $\mathcal{C}_1[\mathcal{C}_2[\cdot]] \triangleq (\!|\mathcal{D}|\!)\big[\ \cdot\ \big]$. By definition, we have that M is a partial state of N. □

The following Lemma relates the behaviour (both the barbs and the reductions) of a partial state M of N to the behaviour of the encoding of N.

Lemma 4.7 *Let M be a partial state of N.*

(1) If $N \downarrow l$ then $M \Downarrow l$; moreover, if $N \xrightarrow{\tau} N'$ then $M \Rightarrow (\!|N'|\!)$.
(2) If $M \downarrow l$ then $N \downarrow l$.
(3) If $M \xrightarrow{\tau} M'$, then $N \xrightarrow{\hat{\tau}} N'$ for some N' such that $M' \gtrsim^{\mathrm{tr}}_{cK} M''$, where M'' is a partial state of N'.

Proof:

(1) By exploiting Proposition 4.2.3 and Lemma 4.3 respectively, this case is simple, once noticed that $M \Rightarrow \gtrsim_{cK} (\!|N|\!)$ (by definition of partial states).
(2) By definition, $N \equiv (\widetilde{\nu l})(N_1 \parallel \cdots \parallel N_n \parallel \overline{N})$ and $M \equiv (\widetilde{\nu l})(M_1 \parallel \cdots \parallel M_n \parallel (\!|\overline{N}|\!))$, where M_i is a partial reduct of N_i. By construction, we know that M_i can only host data on restricted locations; thus, $M_i \downarrow l$ cannot hold. This implies that $(\!|\overline{N}|\!) \downarrow l$ and $l \notin \widetilde{l}$; because of Proposition 4.2.2, $\overline{N} \downarrow l$ and hence $N \downarrow l$.
(3) We know that $N \equiv (\widetilde{\nu l})(N_1 \parallel \cdots \parallel N_n \parallel \overline{N})$, $M \equiv (\widetilde{\nu l})(M_1 \parallel \cdots \parallel M_n \parallel (\!|\overline{N}|\!))$, and for all $i = 1, \ldots, n$ it holds that $fl(N_i) \subseteq fl(\overline{N})$ and that M_i is a partial reduct of N_i. The crucial observation is that there are only two possible cases:
$M_i \xrightarrow{\tau} M_i'$ **and** $M' \equiv (\widetilde{\nu l})(M_1 \parallel \cdots \parallel M_{i-1} \parallel M_i' \parallel M_{i+1} \parallel \cdots \parallel M_n \parallel (\!|\overline{N}|\!))$**.**
By Lemma 4.5 we have three possible sub-cases:
(a) M_i' is a partial reduct of N_i: in this case, M' is still a partial state of N. By construction, $(N, M') \in \mathfrak{R}$.
(b) $M_i' \gtrsim_{cK} (\!|N_i|\!)$: by contextuality of $\lesssim_{cK}$, it holds that $M' \gtrsim_{cK} (\widetilde{\nu l})(M_1 \parallel \cdots \parallel M_{i-1} \parallel M_{i+1} \parallel \cdots \parallel M_n \parallel (\!|N_i \parallel \overline{N}|\!)) \triangleq M''$. Now, M'' is a partial state of N and, hence, $(N, M') \in \mathfrak{R}$ up-to $\lesssim_{\mu K}$.
(c) $N_i \equiv l_1 :: \mathbf{in}(T)@l_2.P \parallel l_2 :: \langle t \rangle$ and $M_i' \gtrsim_{cK} l_1 :: (\!|P\sigma|\!) \parallel l_2 :: \mathbf{nil} \triangleq N'$, where $\sigma = match(T, t)$: in this case, we can consider $N_i \xrightarrow{\tau} l_1 :: P\sigma \parallel l_2 :: \mathbf{nil} \triangleq N_i'$ and have that $M_i' \gtrsim_{cK} (\!|N'|\!)$. Thus, $N \xrightarrow{\tau} (\widetilde{\nu l})(N_1 \parallel N_{i-1} \parallel N_{i+1} \parallel \cdots \parallel N_n \parallel N_i' \parallel \overline{N})$ and $M' \gtrsim_{cK} (\widetilde{\nu l})(M_1 \parallel \cdots \parallel M_{i-1} \parallel M_{i+1} \parallel \cdots \parallel M_n \parallel (\!|N_i' \parallel \overline{N}|\!)) \triangleq M''$. Since M'' is a partial state for N', we have

that $(N', M') \in \mathfrak{R}$ up-to $\lesssim_{\mu K}$.

$\langle\!|\bar{N}|\!\rangle \xrightarrow{\tau} \bar{M}$ **and** $M' \equiv (\widetilde{\nu l})(M_1 \parallel \cdots \parallel M_n \parallel \bar{M})$**.**

By Lemma 4.6, we have two possible sub-cases:

(a) $\bar{N} \xrightarrow{\tau} \bar{N'}$ and $\bar{M} \gtrsim_{cK} \bar{N'}$: in this case, $N \xrightarrow{\tau} (\widetilde{\nu l})(N_1 \parallel \cdots \parallel N_n \parallel \bar{N'}) \triangleq N'$ and $M' \gtrsim_{cK} (\widetilde{\nu l})(M_1 \parallel \cdots \parallel M_n \parallel \langle\!|\bar{N'}|\!\rangle)$; thus, $(N', M') \in \mathfrak{R}$ up-to $\lesssim_{\mu K}$.

(b) $\bar{M}$ is a partial state of $\bar{N}$: by definition, we have that $\bar{N} \equiv (\widetilde{\nu l'})(H_1 \parallel \cdots \parallel H_h \parallel \bar{H})$, $\bar{M} \equiv (\widetilde{\nu l'})(K_1 \parallel \cdots \parallel K_h \parallel \langle\!|\bar{H}|\!\rangle)$, and for all $j = 1, \ldots, h$ it holds that $fl(H_j) \subseteq fl(H)$ and that K_j is a partial reduct of H_j. Thus, $N \equiv (\widetilde{\nu l}, l')(N_1 \parallel \cdots \parallel N_n \parallel H_1 \parallel \cdots \parallel H_h \parallel \bar{H})$ and $M' \equiv (\widetilde{\nu l}, l')(M_1 \parallel \cdots \parallel M_n \parallel K_1 \parallel \cdots \parallel K_h \parallel \langle\!|\bar{H}|\!\rangle)$, where M_i is a partial reduct of N_i and K_j is a partial reduct of H_j. Moreover, we also have that $fl(H_j) \subseteq fl(\bar{H})$ (by definition) and that $fl(N_i) \subseteq fl(\bar{H})$ (this easily follows from $fl(N_i) \subseteq fl(\bar{N})$ and by definition of $\bar{N}$). Thus, M' is a partial state of N; this suffices to conclude $(N, M') \in \mathfrak{R}$. $\square$

To conclude this section, we can formulate a limited full abstraction result, by following [5]. In particular, we shall consider for full abstraction the translated barbed congruence.

Theorem 4.8 (Full Abstraction w.r.t. Translated Barbed Congruence)
$N \cong_{\mu K} M$ if and only if $\langle\!|N|\!\rangle \cong^{\mathrm{tr}}_{cK} \langle\!|M|\!\rangle$.

Proof: For the 'if' direction, it suffices to prove that relation

$$\mathfrak{R} \triangleq \mathfrak{R}_1 \cup \mathfrak{R}_2 \cup \mathfrak{R}_3$$

where

$$\mathfrak{R}_1 \triangleq \{(N, M) : \langle\!|N|\!\rangle \cong_{cK} \langle\!|M|\!\rangle\}$$

$$\mathfrak{R}_2 \triangleq \{(N, M) : \exists \bar{M}.\ \langle\!|N|\!\rangle \cong_{cK} \bar{M} \ \wedge\ \bar{M} \text{ partial state of } M\}$$
$$\mathfrak{R}_3 \triangleq \{(N, M) : \exists \bar{N}.\ \bar{N} \cong_{cK} \langle\!|M|\!\rangle \ \wedge\ \bar{N} \text{ partial state of } N\}$$

is barb preserving, reduction closed (up-to $\lesssim^{\mathrm{tr}}_{cK}$) and closed under translated contexts (again, up-to $\lesssim^{\mathrm{tr}}_{cK}$). Notice that $\mathfrak{R}_1$ is symmetric, while $\mathfrak{R}_2$ and $\mathfrak{R}_3$ are mutually symmetric; thus, $\mathfrak{R}$ is symmetric. We pick up $(N, M) \in \mathfrak{R}$ and reason by case analysis on whether $(N, M) \in \mathfrak{R}_1$, $(N, M) \in \mathfrak{R}_2$ or $(N, M) \in \mathfrak{R}_3$.

(1) Let $N \downarrow l$ (the case for $M \downarrow l$ is similar). By definition and Proposition 4.2.2, we have that $\langle\!|N|\!\rangle \downarrow l$ that implies $\langle\!|M|\!\rangle \Downarrow l$, i.e. $\langle\!|M|\!\rangle \longmapsto^* M' \downarrow l$. According to Lemma 4.6, we have two possibilities:

(a) $M \longmapsto^* M''$ and $M' \gtrsim^{\mathrm{tr}}_{cK} \langle\!|M''|\!\rangle$. In this case, by definition of $\gtrsim^{\mathrm{tr}}_{cK}$, we have that $\langle\!|M''|\!\rangle \downarrow l$ and hence $M \Downarrow l$.

(b) M' is a partial reduct of M. By Lemma 4.7.2, $M \downarrow l$.

Now, let $N \longmapsto N'$; then, $\langle\!|N|\!\rangle \longmapsto^* \bar{N} \gtrsim^{\mathrm{tr}}_{cK} \langle\!|N'|\!\rangle$. By definition of reduction closure, $\langle\!|M|\!\rangle \longmapsto^* \bar{M}$ and $\bar{N} \cong^{\mathrm{tr}}_{cK} \bar{M}$. According to Lemma 4.6, we have two

possibilities:

(a) $M \longmapsto^* M'$ and $\overline{M} \gtrsim^{\mathtt{tr}}_{cK} \langle\!|M'|\!\rangle$. In this case is simple: because of Proposition 2.12 and by transitivity, we can obtain $\langle\!|N'|\!\rangle \cong^{\mathtt{tr}}_{cK} \langle\!|M'|\!\rangle$ and, hence, $(N', M') \in \mathfrak{R}_1$.

(b) M' is a partial reduct of M. By construction, $(N', M) \in \mathfrak{R}_2$ (notice that, if the starting move was from M instead of being from N, the inclusion would have been in $\mathfrak{R}_3$).

We are left with context closure; this case is simple because, if we take any μKLAIM context $\mathcal{C}[\cdot]$, by definition of $\mathfrak{R}_1$ and because $\langle\!|\mathcal{C}|\!\rangle\big[\langle\!|\cdot|\!\rangle\big] = \langle\!|\mathcal{C}[\cdot]|\!\rangle$, we have that $(\mathcal{C}[N], \mathcal{C}[M]) \in \mathfrak{R}_1$.

(2) Let $(N, M) \in \mathfrak{R}_2$; by definition, there exists a partial reduct of M, $\overline{M}$, such that $\langle\!|N|\!\rangle \cong_{cK} \overline{M}$. Let us start with $N \downarrow l$; hence, $\overline{M} \Downarrow l$, i.e. $\overline{M} \longmapsto^* \overline{M}' \downarrow l$. Now, by using Lemma 4.7.3, we have that $M \longmapsto^* M'$ for some M' such that $\overline{M}' \gtrsim^{\mathtt{tr}}_{cK} \overline{M}''$, where $\overline{M}''$ is a partial state of M'. By definition of $\lesssim^{\mathtt{tr}}_{cK}$, we have that $\overline{M}'' \downarrow l$ and, by Lemma 4.7.2, $M' \downarrow l$; this suffices to conclude $M \Downarrow l$.
Now, let $N \longmapsto N'$; then, by Lemma 4.3, $\langle\!|N|\!\rangle \longmapsto^* \overline{N} \gtrsim^{\mathtt{tr}}_{cK} \langle\!|N'|\!\rangle$. By reduction closure, $\overline{M} \longmapsto^* \overline{M}'$ and $\overline{N} \cong^{\mathtt{tr}}_{cK} \overline{M}'$, that implies $\langle\!|N'|\!\rangle \cong^{\mathtt{tr}}_{cK} \overline{M}'$. By Lemma 4.7.3, $M \longmapsto^* M'$ and $\overline{M}'$ is an expansion of a partial state of M', say $\overline{M}''$. By Proposition 2.12 and transitivity, $\langle\!|N'|\!\rangle \cong^{\mathtt{tr}}_{cK} \overline{M}''$; this suffices to conclude that $(N', M') \in \mathfrak{R}_2$.
We are left with context closure; by definition, we have that $\langle\!|\mathcal{C}|\!\rangle\big[\overline{M}\big] \cong^{\mathtt{tr}}_{cK} \langle\!|\mathcal{C}[N]|\!\rangle$. If we prove that $\langle\!|\mathcal{C}|\!\rangle\big[\overline{M}\big]$ is a partial state of $\mathcal{C}[M]$, we can conclude the desired $(\mathcal{C}[N], \mathcal{C}[M]) \in \mathfrak{R}_2$. Since $\overline{M}$ is a partial state of M, we have that $M \equiv (\nu\widetilde{l})(M_1 \parallel \cdots \parallel M_n \parallel \hat{M})$, $\overline{M} \equiv (\nu\widetilde{l})(\overline{M}_1 \parallel \cdots \parallel \overline{M}_n \parallel \langle\!|\hat{M}|\!\rangle)$, and for all $i = 1, \ldots, n$ it holds that $fl(\overline{M}_i) \subseteq fl(\hat{M})$ and that $\overline{M}_i$ is a partial reduct of M_i. Let $\widetilde{l'} = bn(\langle\!|\mathcal{C}|\!\rangle\big[\,\cdot\,\big]) \cap fn(\overline{M}_1, \ldots, \overline{M}_n) = bn(\mathcal{C}[\cdot]) \cap fn(M_1, \ldots, M_n)$; then, $\mathcal{C}[\cdot] \equiv (\nu\widetilde{l'})\mathcal{D}[\cdot]$ and $\langle\!|\mathcal{C}|\!\rangle\big[\,\cdot\,\big] \equiv (\nu\widetilde{l'})\langle\!|\mathcal{D}|\!\rangle\big[\,\cdot\,\big]$. Thus, $\mathcal{C}[M] \equiv (\nu\widetilde{l'})(M_1 \parallel \cdots \parallel M_n \parallel \mathcal{D}[\hat{M}])$ and $\langle\!|\mathcal{C}|\!\rangle\big[\overline{M}\big] \equiv (\nu\widetilde{l'})(\overline{M}_1 \parallel \cdots \parallel \overline{M}_n \parallel \langle\!|\mathcal{D}|\!\rangle\big[\overline{M}\big])$; clearly, $\langle\!|\mathcal{C}|\!\rangle\big[\overline{M}\big]$ is a partial state of $\mathcal{C}[M]$.

(3) Finally, let $(N, M) \in \mathfrak{R}_3$; by definition, there exists a partial reduct of N, $\overline{N}$, such that $\langle\!|M|\!\rangle \cong_{cK} \overline{N}$. Let us start with barb preservation and let $N \downarrow l$; by Lemma 4.7.1, $\overline{N} \Downarrow l$, i.e. $\overline{N} \longmapsto^* \overline{N}' \downarrow l$. Now, $\langle\!|M|\!\rangle \Downarrow l$ that, like in case 1. above, implies $M \Downarrow l$, as required.
Now, let $N \longmapsto N'$; then, Lemma 4.7.1, $\overline{N} \longmapsto^* \overline{N}' \gtrsim^{\mathtt{tr}}_{cK} \langle\!|N'|\!\rangle$. By reduction closure, $\langle\!|M|\!\rangle \longmapsto^* \overline{M}$ and $\overline{N}' \cong^{\mathtt{tr}}_{cK} \overline{M}$, that implies $\langle\!|N'|\!\rangle \cong^{\mathtt{tr}}_{cK} \overline{M}$. By Lemma 4.6, we have two possibilities:

(a) $M \longmapsto^* M'$ and $\overline{M} \gtrsim^{\mathtt{tr}}_{cK} \langle\!|M'|\!\rangle$. By Proposition 2.12 and transitivity, we can conclude that $(N', M') \in \mathfrak{R}_1$.

(b) $M \longmapsto^* M'$ and $\overline{M}$ is a partial state of M'. By construction, $(N', M') \in \mathfrak{R}_2$.

Context closure is proved like in case 2. above.

We are left with the 'only if' direction; this can be done similarly to the 'if' direction. We leave the details to the reader. □

Encoding Nets:

$$\llbracket \mathbf{0} \rrbracket \triangleq \mathbf{0} \qquad \llbracket (\nu l)N \rrbracket \triangleq (\nu l)\llbracket N \rrbracket$$

$$\llbracket N_1 \parallel N_2 \rrbracket \triangleq \llbracket N_1 \rrbracket \parallel \llbracket N_2 \rrbracket \qquad \llbracket l :: C \rrbracket \triangleq l :: \llbracket C \rrbracket_l$$

Encoding Components:

$$\llbracket \langle l' \rangle \rrbracket_u \triangleq \langle l' \rangle \qquad \llbracket C_1 \mid C_2 \rrbracket_u \triangleq \llbracket C_1 \rrbracket_u \mid \llbracket C_2 \rrbracket_u$$

$$\llbracket \mathbf{nil} \rrbracket_u \triangleq \mathbf{nil} \qquad \llbracket X \rrbracket_u \triangleq X$$

$$\llbracket \mathbf{eval}(Q)@u'.P \rrbracket_u \triangleq \mathbf{eval}(\llbracket Q \rrbracket_{u'})@u'.\llbracket P \rrbracket_u \qquad \llbracket \mathbf{rec}\, X.P \rrbracket_u \triangleq \mathbf{rec}\, X.\llbracket P \rrbracket_u$$

$$\llbracket \mathbf{out}(u_2)@u_1.P \rrbracket_u \triangleq \mathbf{eval}(\mathbf{out}(u_2))@u_1.\llbracket P \rrbracket_u \qquad \llbracket \mathbf{new}(l).P \rrbracket_u \triangleq \mathbf{new}(l).\llbracket P \rrbracket_u$$

$$\llbracket \mathbf{in}(T)@u'.P \rrbracket_u \triangleq \mathbf{eval}\big(\mathbf{in}(T).\mathbf{eval}(\llbracket P \rrbracket_u)@u\big)@u'$$

Table 12
Encoding cKLAIM in lcKLAIM

5 cKLAIM vs lcKLAIM

In this section we develop a semantically equivalent encoding of cKLAIM in lcKLAIM. To the best of our knowledge, this is the first result that clearly shows that remote (input and output) operations do not add expressiveness to a distributed language with code mobility. Indeed, we have that the possibility of using remote operations simplifies programming, however a calculus with migrations as the only remote operation permits finer dynamic checks against incoming agents (see, e.g., [28,19]).

The encoding of cKLAIM into lcKLAIM is given in Table 12. The only relevant cases are those for the translation of actions **in** and **out** of cKLAIM. In the first case, a remote action **out** is replaced with a migration to the target locality and a local action. In the second case, a remote action **in** is replaced with a migration to the target locality, a local **in** and a migration back to the original node. The subscript u in $\llbracket \cdot \rrbracket_u$ is needed to keep track of the original node where report the result of the (remote) action.

In order to carry on the proofs, we introduce an auxiliary notion. We define a function between lcKLAIM nets $nrm_L(\cdot)$, called the *normalization w.r.t. a set of localities*

L, as follows

$$nrm_L(N_1 \parallel N_2) \triangleq nrm_L(N_1) \parallel nrm_L(N_2) \qquad nrm_L((\nu l)N) \triangleq nrm_{L\cup\{l\}}(N)$$

$$nrm_L(l :: C_1 \mid C_2) \triangleq nrm_L(l :: C_1) \parallel nrm_L(l :: C_2) \qquad nrm_L(l :: \langle\cdot\rangle) \triangleq l :: \langle\cdot\rangle$$

$$nrm_L(l :: P) \triangleq \begin{cases} l :: P' \parallel l' :: Q & \text{if } P = a.P' \text{ and } a = \mathbf{eval}(Q)@l' \text{ and} \\ & \quad l' \in L \text{ and } Q = \mathbf{in}(T).\mathbf{eval}(\{\!|P|\!\}_l)@l \\ l :: P & \text{if } P \neq \_ \mid \_ \end{cases}$$

Essentially, the normalisation of an encoding replaces all the encodings of actions **in** occurring at top level (i.e., as the first action of a process) with the net resulting from the execution of their first actions (i.e., the migration over the locality target of the **in**), provided that this execution is possible (i.e., the target locality of the input exists in the net).

Now, it is easy to prove the following Proposition. For the sake of readability, we write $nrm_L(\{\!|N|\!\})$ as $\{\!|N|\!\}_L$ and $\{\!|N|\!\}_{fl(N)}$ as $\{\!|N|\!\}$.

Proposition 5.1 *Let N be a* cKlaim *net and M be a* lcKlaim *net. Then*

(1) $fl(M) = fl(\{\!|M|\!\}_L)$, *whenever* $L \subseteq fl(M)$
(2) $M \gtrsim_{lcK} nrm_L(M)$, *whenever* $L \subseteq fl(M)$
(3) $\{\!|N|\!\}_L \parallel l :: \mathbf{nil} \gtrsim_{lcK} \{\!|N|\!\}_{L\cup\{l\}} \parallel l :: \mathbf{nil}$
(4) $\{\!|N|\!\} \gtrsim_{lcK} \{\!|N|\!\}$

Lemma 5.2 *Let N be a* cKlaim *net and $fl(N) \subseteq L$. Then*

(1) if $N \xrightarrow{(\nu\widetilde{l})\ I\ @\ l} N'$ *then* $\{\!|N|\!\}_L \xRightarrow{(\nu\widetilde{l})\ I\ @\ l} \{\!|N'|\!\}_{L\cup\{\widetilde{l}\}}$
(2) if $N \xrightarrow{\triangleright\, l} N'$ *then* $\{\!|N|\!\}_L \xRightarrow{\triangleright\, l} \{\!|N'|\!\}_L$
(3) if $N \xrightarrow{l_2 \,\triangleleft\, l_1} N'$ *then either* $\{\!|N|\!\}_L \xRightarrow{l_2 \,\triangleleft\, l_1} \{\!|N'|\!\}_L$, *or* $\{\!|N|\!\}_L \equiv \mathcal{C}[l :: \mathbf{eval}(\mathbf{in}(T).\mathbf{eval}(\{\!|P|\!\}_l)@l)@l_1]$ *where* $match(T, l_2) = \sigma$, $l_1 \notin L$, $\{l_1, l_2\} \cap bn(\mathcal{C}[\cdot]) = \emptyset$ *and* $\{\!|N'|\!\}_L \equiv nrm_L(\mathcal{C}[l :: \{\!|P\sigma|\!\}_l])$
(4) if $\{\!|N|\!\}_L \xrightarrow{(\nu\widetilde{l})\ I\ @\ l} N'$, *then* $N \xrightarrow{(\nu\widetilde{l})\ I\ @\ l} N''$ *and* $N' \gtrsim_{lcK} \{\!|N''|\!\}_{L\cup\{\widetilde{l}\}}$
(5) if $\{\!|N|\!\}_L \xrightarrow{l_2 \,\triangleleft\, l_1} N'$, *then* $N \xrightarrow{l_2 \,\triangleleft\, l_1} N''$ *and* $N' \gtrsim_{lcK} \{\!|N''|\!\}_L$
(6) if $\{\!|N|\!\}_L \xrightarrow{\triangleright\, l} N'$ *then*
  (a) either $N \xrightarrow{\triangleright\, l} N''$ *and* $N' \gtrsim_{lcK} \{\!|N''|\!\}_L$
  (b) or $N \equiv \mathcal{C}[l' :: \mathbf{in}(T)@l.Q]$ *for* $l \notin bn(\mathcal{C}[\cdot]) \cup L$ *and* $N' \equiv \{\!|\ \mathcal{C}[l' :: \mathbf{nil} \parallel l :: \mathbf{in}(T).\mathbf{eval}(\{\!|P|\!\}_l)@l]\ |\!\}_L$

Proof: All the statements are proved by induction on the length of the inference used to derive the transition; the proof is standard. □

Lemma 5.3 (Operational Correspondence) *Let N be a* cKlaim *net. Then*

(1) if $N \xrightarrow{\tau} N'$, then $\{\!|N|\!\} \Rightarrow \gtrsim_{lcK} \{\!|N'|\!\}$
(2) if $\{\!|N|\!\} \xrightarrow{\tau} N'$, then $N \xrightarrow{\tau} N''$ and $N' \gtrsim_{lcK} \{\!|N''|\!\}$

Proof: Both the claims are proved by induction on the inference length. The inductive steps are easy: they rely on the fact that $\lesssim_{lcK}$ is a pre-congruence and on the observation that $N \equiv M$ implies $\{\!|N|\!\} \equiv \{\!|M|\!\}$. Thus, we only give the base cases for both the claims.

In the first case, the τ-step can be inferred by using rules (LTS-New), (LTS-Send) or (LTS-Comm). The first case is simple; hence, let us consider the other two.

(LTS-Send**):** in this case, $N \triangleq N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N'$, where $N_1 \xrightarrow{\triangleright\, l} N_1'$ and $N_2 \xrightarrow{\mathbf{nil}\ @\ l} N_2'$. The key observation is that $fl(N_i) \subseteq fl(N) = fl(N_1' \parallel N_2') = fl(N')$; let us call L the set $fl(N)$. By Lemmata 5.2.1 and .2, we have that $\{\!|N_2|\!\}_L \xRightarrow{\mathbf{nil}\ @\ l} \{\!|N_2'|\!\}_L$ and $\{\!|N_1|\!\}_L \xrightarrow{\triangleright\, l} \{\!|N_1'|\!\}_L$. Thus, $\{\!|N|\!\} \Rightarrow \{\!|N_1' \parallel N_2'|\!\} \triangleq \{\!|N'|\!\}$.

(LTS-Comm**):** in this case, $N \triangleq N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N'$, where $N_1 \xrightarrow{l_2 \triangleleft l_1} N_1'$ and $N_2 \xrightarrow{\langle l_2 \rangle\ @\ l_1} N_2'$. Again, we have that $fl(N_i) \subseteq fl(N) = fl(N')$; let us call L the set $fl(N)$. By Lemma 5.2.1 we have that $\{\!|N_2|\!\}_L \xRightarrow{\langle l_2 \rangle\ @\ l_1} \{\!|N_2'|\!\}_L$. Moreover, according to Lemma 5.2.3, we have two cases. The case for $\{\!|N_1|\!\}_L \xRightarrow{l_2 \triangleleft l_1} \{\!|N_1'|\!\}_L$ is simple. The case when $\{\!|N_1|\!\}_L \equiv \mathcal{C}[l :: \mathbf{eval}(\mathbf{in}(T).\mathbf{eval}(\{\!|P|\!\}_l)@l)@l_1]$ cannot occur. Otherwise, we would have that $l_1 \notin L$; but this cannot be the case since, by Proposition 2.8.2, we know that $N_2 \equiv N_2' \parallel l_1 :: \langle l_2 \rangle$. Hence $l_1 \in fl(N_2) \subseteq fl(N) \triangleq L$.

The second claim is similar. We reason by case analysis on the possible base cases. The case for rule (LTS-New) is simple and we only inspect the other two. In what follows, we let L to be $fl(N)$.

(LTS-Send**):** in this case, $\{\!|N|\!\} \triangleq M_1 \parallel M_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N'$, where $M_1 \triangleq \{\!|N_1|\!\}_L \xrightarrow{\triangleright\, l} N_1'$ and $M_2 \triangleq \{\!|N_2|\!\}_L \xrightarrow{\mathbf{nil}\ @\ l} N_2'$. By Lemma 5.2.1 we have that $N_2 \xrightarrow{\mathbf{nil}\ @\ l} N_2''$ and $N_2' \gtrsim_{lcK} \{\!|N_2''|\!\}_L$. We now isolate two sub-cases:

(a) $C = \{\!|P|\!\}_l$. Then, by Lemma 5.2.6(a) we have that $N_1 \xrightarrow{\triangleright\, l} N_1''$ and $N_1' \gtrsim_{lcK} \{\!|N_1''|\!\}_L$. Thus, $N \triangleq N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N''$ and $N' \gtrsim_{lcK} \{\!|N_1'' \parallel N_2''|\!\} \triangleq \{\!|N''|\!\}$.

(b) $C \triangleq \mathbf{in}(T).\mathbf{eval}(\{\!|Q|\!\}_{l'})@l'$. By Lemma 5.2.6(b) we know that $N_1 \equiv \mathcal{C}[l' :: \mathbf{in}(T)@l.Q]$, with $l \notin bn(\mathcal{C}[\cdot]) \cap L$. This case cannot occur because, by By Proposition 2.8.1, we know that $N_2 \equiv N_2' \parallel l :: \mathbf{nil}$; hence, $l \in L$.

(LTS-Comm**):** in this case, $\{\!|N|\!\} \triangleq M_1 \parallel M_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N'$, where $M_1 \triangleq \{\!|N_1|\!\}_L \xrightarrow{l_2 \triangleleft l_1} N_1'$ and $M_2 \triangleq \{\!|N_2|\!\}_L \xrightarrow{\langle l_2 \rangle\ @\ l_1} N_2'$. By Lemma 5.2.4 we have that

$N_2 \xrightarrow{\langle l_2\rangle\ @\ l_1} N_2''$ and $N_2' \gtrsim_{lcK} \{\!|N_2''|\!\}_L$; by Lemma 5.2.5 we have that $N_1 \xrightarrow{l_2 \triangleleft l_1} N_1''$ and $N_1' \gtrsim_{lcK} \{\!|N_1''|\!\}_L$. Thus, $N \triangleq N_1 \parallel N_2 \xrightarrow{\tau} N_1'' \parallel N_2'' \triangleq N''$ and $N' \gtrsim_{lcK} \{\!|N_1''|\!\}_L \parallel \{\!|N_2''|\!\}_L \triangleq \{\!|N''|\!\}$. □

Theorem 5.4 *Let N be a* cKlaim *net. Then, $N \cong_{cK} \{\!|N|\!\}$.*

Proof: By Lemma 2.15.1, it suffices to prove that

$$\mathfrak{R} \triangleq \{(\mathcal{C}[N], \mathcal{C}[\{\!|N|\!\}]) \ : \ N \text{ is a cKlaim net and } \mathcal{C}[\cdot] \text{ is a cKlaim context}\}$$

is barb preserving, reduction closed (up-to $\precsim_{cK}$) and context closed. Clearly, we consider here the restriction of $\cong_{\mu K}$ and $\precsim_{\mu K}$ to cKlaim nets; all the proofs developed in Section 2.6 for μKlaim can be faithfully rephrased to deal with the subrelations containing only cKlaim nets.

Barb preservation and context closure are simple. Let us consider $\mathcal{C}[N] \longmapsto \bar{N}$. According to Lemma 2.9, we have six possible sub-cases:

(1) $N \longmapsto N'$ and $\bar{N} \equiv \mathcal{C}[N']$. Because of Lemma 5.3.1, we know that $\{\!|N|\!\} \Rightarrow \gtrsim_{cK} \{\!|N'|\!\}$; thus, we can conclude up-to $\precsim_{cK}$.
(2) $\mathcal{C}[\cdot] \longmapsto \mathcal{C}'[\cdot]$ and $\bar{N} \equiv \mathcal{C}'[N]$. This case is trivial.
(3) $N \xrightarrow{\triangleright\, l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}[\ \cdot \parallel l :: \mathbf{nil}]$ and $\bar{N} \equiv \mathcal{C}[N']$. Because of Lemma 5.2.2, we know that $\{\!|N|\!\} \stackrel{\triangleright\, l}{\Longrightarrow} \gtrsim_{cK} \{\!|N'|\!\}$ and we can easily conclude.
(4) $N \xrightarrow{\mathbf{nil}\ @\ l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\ \cdot \parallel H]$, $H \xrightarrow{\triangleright\, l} H'$ and $\bar{N} \equiv \mathcal{C}'[N' \parallel L']$. This case relies on Lemma 5.2.1 and is simple.
(5) $N \xrightarrow{l' \triangleleft l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\ \cdot \parallel l :: \langle l'\rangle]$ and $\bar{N} \equiv \mathcal{C}'[N']$. The proof relies on Lemma 5.2.3 to show that $\mathcal{C}[\{\!|N|\!\}] \Rightarrow \mathcal{C}'[\{\!|N'|\!\}]$; now it is easy to conclude.
(6) $N \xrightarrow{(\nu \widetilde{l})\ \langle l'\rangle\ @\ l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\ \cdot \parallel H]$, $H \xrightarrow{l' \triangleleft l} H'$ and $\bar{N} \equiv \mathcal{C}'[(\nu \widetilde{l})(N' \parallel H')]$. This case relies on Lemma 5.2.1 and is simple.

To conclude, let us consider $\mathcal{C}[\{\!|N|\!\}] \longmapsto \bar{N}$. According to Lemma 2.9, we still have six possible sub-cases:

(1) $\{\!|N|\!\} \longmapsto N'$ and $\bar{N} \equiv \mathcal{C}[N']$. Because of Lemma 5.3.3, we know that $N \longmapsto N''$ and $N' \gtrsim_{lcK} \{\!|N|\!\}$; this suffices to conclude up-to $\precsim_{cK}$ (indeed, by considering both N' and $\{\!|N|\!\}$ as cKlaim nets, we have that $N' \gtrsim_{cK} \{\!|N|\!\}$).
(2) $\mathcal{C}[\cdot] \longmapsto \mathcal{C}'[\cdot]$ and $\bar{N} \equiv \mathcal{C}'[\{\!|N|\!\}]$. This case is trivial.
(3) $\{\!|N|\!\} \xrightarrow{\triangleright\, l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}[\ \cdot \parallel l :: \mathbf{nil}]$ and $\bar{N} \equiv \mathcal{C}[N']$. Because of Lemma 5.2.6, we have two possible sub-cases:
   (a) $N \xrightarrow{\triangleright\, l} N''$ and $N' \gtrsim_{cK} \{\!|N''|\!\}$. In this case it is easily to conclude.
   (b) $N \equiv \mathcal{D}[l' :: \mathbf{in}(T)@l.P]$, for $l \notin bn(\mathcal{D}[\cdot]) \cup fn(N)$, and $N' \equiv \{\!|\mathcal{D}|\!\}[l' :: \mathbf{nil} \parallel l :: \mathbf{in}(T).\mathbf{eval}(\{\!|P|\!\}_{l'})@l']$. Now, $\mathcal{C}[\{\!|N|\!\}] \longmapsto \mathcal{C}[\{\!|\mathcal{D}|\!\}[l' :: \mathbf{nil} \parallel l :: \mathbf{in}(T).\mathbf{eval}(\{\!|P|\!\}_{l'})@l']] \triangleq$

$$\frac{p =_\alpha p' \xrightarrow{\mu} q' =_\alpha q}{p \xrightarrow{\mu} q} \qquad a(b).p \xrightarrow{ac} p[c/b] \qquad \bar{a}ab \xrightarrow{\bar{a}b} \mathbf{0} \qquad \frac{p \xrightarrow{\bar{a}b} p' \qquad q \xrightarrow{ab} q'}{p \mid q \xrightarrow{\tau} p' \mid q'}\ (*)$$

$$\frac{p \xrightarrow{\mu} p' \qquad a \notin fn(\mu)}{(\nu b)p \xrightarrow{\mu} (\nu b)p'} \qquad \frac{p \xrightarrow{\bar{a}b} p' \qquad a \neq b}{(\nu b)p \xrightarrow{\bar{a}(b)} p'} \qquad \frac{p \xrightarrow{\bar{a}(b)} p' \qquad q \xrightarrow{ab} q' \qquad b \notin fn(q)}{p \mid q \xrightarrow{\tau} (\nu b)(p' \mid q')}\ (*)$$

$$\frac{p \xrightarrow{\mu} p'}{[a = a]p \xrightarrow{\mu} p'} \qquad \frac{p \xrightarrow{\mu} p'}{!p \xrightarrow{\mu} !p \mid p'} \qquad \frac{p \xrightarrow{\mu} p' \qquad bn(\mu) \cap fn(q) = \emptyset}{p \mid q \xrightarrow{\mu} p' \mid q}\ (*)$$

and the symmetric versions of the rules marked with (*)

Table 13
A LTS for π_a-calculus

$\bar{N} \gtrsim_{cK} \mathcal{C}[\{\!|\mathcal{D}[l' :: \mathbf{nil} \parallel l :: \mathbf{in}(T).\mathbf{eval}(\{\!|P|\!\}_{l'})@l']|\!\}] \triangleq \mathcal{C}[\{\!|N \parallel l :: \mathbf{nil}|\!\}]$ (the last inequality holds by Proposition 5.1.3). Now, since $\mathcal{C}[N] \equiv \mathcal{C}[N \parallel l :: \mathbf{nil}]$, we have that $(\mathcal{C}[N], \bar{N}) \in \mathfrak{R}$ up-to $\lesssim_{cK}$, as required.

(4) $\{\!|N|\!\} \xrightarrow{\mathbf{nil}\ @\ l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\ \cdot \parallel H]$, $H \xrightarrow{\triangleright\, l} H'$ and $\bar{N} \equiv \mathcal{C}'[N' \parallel L']$. This case relies on Lemma 5.2.4 and is simple.

(5) $\{\!|N|\!\} \xrightarrow{l' \triangleleft l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\ \cdot \parallel l :: \langle l' \rangle]$ and $\bar{N} \equiv \mathcal{C}'[N']$. The proof relies on Lemma 5.2.5 and is simple.

(6) $\{\!|N|\!\} \xrightarrow{(\nu \widetilde{l})\ \langle l' \rangle\ @\ l} N'$, $\mathcal{C}[\cdot] \equiv \mathcal{C}'[\ \cdot \parallel H]$, $H \xrightarrow{l' \triangleleft l} H'$ and $\bar{N} \equiv \mathcal{C}'[(\nu \widetilde{l})(N' \parallel H')]$. This case relies on Lemma 5.2.4 and is simple. □

Corollary 5.5 (Semantical Equivalence w.r.t. $\cong_{cK}$) *Let N be a* cKLAIM *net. Then, $N \cong_{cK} \{\!|N|\!\}$.*

Proof: By Propositions 5.1.4 and 2.12, Theorem 5.4 and by transitivity of $\cong_{cK}$. □

6 A Comparison with π_a-calculus

In this section, we want to compare asynchronous π-calculus, that we write π_a-calculus, with our languages. In particular, we develop a fully abstract and divergence-free encoding of π_a-calculus in cKLAIM and a fully abstract but divergent encoding of lcKLAIM in π_a-calculus.

The variant of π_a-calculus that we consider in this paper is adapted from [1]. Its

syntax is

$$p \;::=\; \mathbf{0} \;\Big|\; \bar{a}b \;\Big|\; a(b).p \;\Big|\; p_1|p_2 \;\Big|\; (\nu a)p \;\Big|\; [a=b]p \;\Big|\; !p$$

while its operational semantics is given in Table 13. On top of that LTS, barbed equivalence is defined as follows (see also [1]).

Definition 6.1 (Asynchronous Barbed Equivalence) Asynchronous barbed equivalence, $\cong_{\pi_a}$, *is the largest symmetric relation between* π_a*-calculus processes such that* $p \cong_{\pi_a} q$ *implies that*

(1) whenever $p \downarrow \bar{a}$*, it holds that* $q \Downarrow \bar{a}$*, where* $p \downarrow \bar{a} \triangleq (\exists b.\, p \xrightarrow{\bar{a}b} \;\vee\; p \xrightarrow{\bar{a}(b)})$ *and* $p \Downarrow \bar{a} \triangleq (p \Rightarrow \downarrow \bar{a})$
(2) whenever $p \xrightarrow{\tau} p'$*, it holds that* $q \Rightarrow q'$ *and* $p' \cong_{\pi_a} q'$
(3) for all names $\widetilde{n}$ *and* π_a*-calculus process* r*, it holds that* $(\nu\widetilde{n})(p|r) \cong_{\pi_a} (\nu\widetilde{n})(q|r)$.

6.1 Encoding π_a-calculus in cKlaim

We now provide an encoding of π_a-calculus in cKlaim; it is given in Table 14. Like in the previous section, we need a normalization function between cKlaim nets that makes the encoding prompt. It is defined as follows:

$$nrm_L((\nu l)N) \triangleq nrm_{L\cup\{l\}}(N) \qquad nrm_L(N_1 \parallel N_2) \triangleq nrm_L(N_1) \parallel nrm_L(N_2)$$

$$nrm_L(l :: \langle\cdot\rangle) \triangleq l :: \langle\cdot\rangle \qquad nrm_L(l :: C_1 \mid C_2) \triangleq nrm_L(l :: C_1) \parallel nrm_L(l :: C_2)$$

$$nrm_L(l :: P) \triangleq \begin{cases} l :: P' \parallel l' :: \langle l''\rangle & \text{if } P = \mathbf{out}(l'')@l'.P' \text{ and } l' \in L \\ (\nu l')\,(nrm_{L\cup\{l'\}}(l :: P')) & \text{if } P = \mathbf{new}(l').P' \\ l :: P & \text{if } P \neq \_|\_ \text{ and no previous case holds} \end{cases}$$

Essentially, the normalisation replaces all actions **new** with the net resulting from the creation of the new nodes and all actions **out** over existing localities with the net containing the datum produced by the action. When a net is the encoding of a π_a-calculus process, the continuation of each action **out** is **nil** and function *nrm* does not need to be iterated on it.

For the sake of readability, we write $nrm_L([\![p]\!]_L)$ as $[\![\![p]\!]\!]_L$. Some simple but crucial properties of $nrm_L(\cdot)$ are given in the following proposition, whose proof is simple.

Proposition 6.2 *Let* P *be a* cKlaim *process and* p *be a* π_a*-calculus process. Then*

(1) if $l \neq l'$ *then* $nrm_L((\nu l')(l :: P)) \;=\; nrm_L(l :: \mathbf{new}(l').P)$

Top-level Encoding:

$$[\![p]\!]_L \triangleq \texttt{proc} :: [\![p]\!] \parallel \prod_{n \in L} n :: \mathbf{nil} \qquad \text{if } fn(p) \subseteq L \text{ and } \texttt{proc} \text{ is a reserved name}$$

Encoding π_a-calculus processes:

$$[\![\mathbf{0}]\!] \triangleq \mathbf{nil} \qquad [\![(\nu a)p]\!] \triangleq \mathbf{new}(a).[\![p]\!]$$

$$[\![\bar{a}b]\!] \triangleq \mathbf{out}(b)@a.\mathbf{nil} \qquad [\![a(b).p]\!] \triangleq \mathbf{in}(!b)@a.[\![p]\!]$$

$$[\![p_1|p_2]\!] \triangleq [\![p_1]\!] \mid [\![p_2]\!] \qquad [\![!p]\!] \triangleq \mathbf{rec}\, X.([\![p]\!] \mid X)$$

$$[\![[a = b]p]\!] \triangleq \mathbf{new}(l).\mathbf{out}(a)@l.\mathbf{in}(b)@l.[\![p]\!]$$

Table 14
Encoding π_a-calculus in cKLAIM

(2) $[\![p]\!]_L \parallel l :: \mathbf{nil} \gtrsim_{cK} [\![p]\!]_{L\cup\{l\}}$
(3) $[\![p]\!]_L \gtrsim_{lcK} [\![p]\!]_L$

We now prove a tight correspondence between π_a-calculus processes and their encodings. We start with a correspondence between the labelled semantics of the two calculi and then give their operational correspondence.

Lemma 6.3 *Let p be a π_a-calculus process and $fn(p) \subseteq L$. Then*

(1) *if* $p \xrightarrow{\bar{a}b} p'$ *then* $[\![p]\!]_L \xRightarrow{\langle b\rangle @ a} [\![p']\!]_L$
(2) *if* $p \xrightarrow{\bar{a}(b)} p'$ *then* $[\![p]\!]_L \xRightarrow{(\nu b)\,\langle b\rangle @ a} [\![p']\!]_{L\cup\{b\}}$
(3) *if* $p \xrightarrow{ab} p'$ *then* $[\![p]\!]_L \xRightarrow{b \triangleleft a} N$ *and* $[\![p']\!]_{L\cup\{b\}} \equiv N \parallel b :: \mathbf{nil}$
(4) *if* $[\![p]\!]_L \xrightarrow{\langle b\rangle @ a} N$ *then* $p \xrightarrow{\bar{a}b} p'$ *and* $N \equiv [\![p']\!]_L$
(5) *if* $[\![p]\!]_L \xrightarrow{(\nu b)\,\langle b\rangle @ a} N$ *then* $p \xrightarrow{\bar{a}(b)} p'$ *and* $[\![p']\!]_{L\cup\{b\}} \equiv N \parallel b :: \mathbf{nil}$
(6) *if* $[\![p]\!]_L \xrightarrow{b \triangleleft a} N$ *then* $p \xrightarrow{ab} p'$ *and* $[\![p']\!]_{L\cup\{b\}} \lesssim_{cK} N \parallel b :: \mathbf{nil}$

Proof: By induction on the length of the inferences. □

Lemma 6.4 (Operational Correspondence) *Let p be a π_a-calculus process and $fn(p) \subseteq L$. Then*

(1) $p \xrightarrow{\tau} p'$ *implies that* $[\![p]\!]_L \Rightarrow [\![p']\!]_L$
(2) $[\![p]\!]_L \xrightarrow{\tau} N$ *implies that* $p \xrightarrow{\tau} p'$ *and* $N \gtrsim_{cK} [\![p']\!]_L$

Proof: Both the claims are proved by induction on the inference occurring in the premise. The first statement is quite simple. We give the base cases for the second

statement. We want to remark that, thanks to the normalization procedure, the only possible base case is when using rule (LTS-Comm). Thus, $[\![p]\!]_L \triangleq N_1 \parallel N_2 \xrightarrow{\tau} N_1' \parallel N_2' \triangleq N$, because $N_1 \xrightarrow{b \triangleleft a} N_1'$ and $N_2 \xrightarrow{\langle b \rangle @ a} N_2'$. By definition, it must be that $N_i \triangleq [\![p_i]\!]_L$, for $i = 1, 2$; moreover, $\{a, b\} \subseteq fn(N_2) \subseteq L$. By Lemmata 6.3.6 and .4, we have that $p_1 \xrightarrow{ab} p_1'$ and $[\![p_1']\!]_L \lesssim_{cK} N_1' \parallel b :: \mathbf{nil}$, and $p_2 \xrightarrow{\overline{ab}} p_2'$ and $N_2' \equiv [\![p_2']\!]_L$. Thus, $p \triangleq p_1|p_2 \xrightarrow{\tau} p_1'|p_2' \triangleq p'$. Moreover, $N \equiv N_1' \parallel b :: \mathbf{nil} \parallel N_2' \gtrsim_{cK} [\![p_1']\!]_L \parallel [\![p_2']\!]_L \triangleq [\![p']\!]_L$, as required. □

We now prove that full abstraction can be obtained when considering only the following subset of cKlaim contexts, called *translated*:

$$\mathcal{C}[\cdot] \quad ::= \quad [\cdot] \quad \Big| \quad \mathcal{C}[\cdot] \parallel [\![p]\!]_L \quad \Big| \quad (\nu l)\mathcal{C}[\cdot]$$

Basically, we only permit parallel components resulting from the encoding of π_a-calculus processes. This ensures that each free name occurring in any parallel component is also the address of a node in the component itself and is essential to prove full abstraction.

Theorem 6.5 *Let $fn(p, q) \subseteq L$. Then $p \cong_{\pi_a} q$ if and only if $[\![p]\!]_L \cong^{\mathtt{tr}}_{cK} [\![q]\!]_L$.*

Proof: We start with proving that $[\![p]\!]_L \cong^{\mathtt{tr}}_{cK} [\![q]\!]_L$ implies $p \cong_{\pi_a} q$. Let $\mathfrak{R} \triangleq \{(p\ ,\ q) : [\![p]\!]_L \cong^{\mathtt{tr}}_{cK} [\![q]\!]_L\}$; we prove that $\mathfrak{R} \subseteq \cong_{\pi_a}$. Let $p\ \mathfrak{R}\ q$. For reduction closure, we take $p \xrightarrow{\tau} p'$; by Lemma 6.4.1 it holds that $[\![p]\!]_L \Rightarrow [\![p']\!]_L$. Thus, $[\![q]\!]_L \Rightarrow N$ and $[\![p']\!]_L \cong^{\mathtt{tr}}_{cK} N$. Then, by using Lemma 6.4.2, it can be easily verified that $q \Rightarrow q'$ and $[\![q']\!]_L \lesssim_{cK} N$. This suffices to conclude that $p'\ \mathfrak{R}\ q'$ since, by the fact that $\lesssim_{cK} \subseteq \cong_{cK} \subseteq \cong^{\mathtt{tr}}_{cK}$ and by transitivity of $\cong^{\mathtt{tr}}_{cK}$, it holds that $[\![p']\!]_L \cong^{\mathtt{tr}}_{cK} [\![q']\!]_L$ (notice that, as it is standard in π-calculus, $fn(p') \subseteq fn(p)$ and, thus, $fn(p') \subseteq L$ – and similarly for q and q').

We now consider barb preservation; let $p \downarrow \overline{a}$ because $p \xrightarrow{\overline{ab}}$. By Lemma 6.3.1 it holds that $[\![p]\!]_L \xrightarrow{b @ a}$; thus, $[\![q]\!]_L \xrightarrow{b @ a}$. Hence, by Lemma 6.3.4 and reduction closure (just proved), it holds that $q \Rightarrow \xrightarrow{\overline{ab}}$; thus, by definition, $q \Downarrow \overline{a}$. The case for $p \downarrow \overline{a}$ because $p \xrightarrow{\overline{a(b)}}$ is similar, but relies on Lemmata 6.3.2 and .5.

Finally, we have to prove closure under parallel composition and restriction. Let us examine the two conditions separately.

- We want to prove that $(\nu\widetilde{n})p\ \mathfrak{R}\ (\nu\widetilde{n})q$ by knowing that $(\nu\widetilde{n})([\![p]\!]_L) \cong^{\mathtt{tr}}_{cK} (\nu\widetilde{n})([\![q]\!]_L)$. By definition, we have that $(\nu\widetilde{n})([\![\cdot]\!]_L) \triangleq [\![(\nu\widetilde{n}) \cdot]\!]_{L-\{\widetilde{n}\}}$ and $fn((\nu\widetilde{n})\ \cdot) \triangleq fn(\cdot) - \{\widetilde{n}\}$. Thus, $fn((\nu\widetilde{n})p, (\nu\widetilde{n})q) \subseteq L - \{\widetilde{n}\}$ and hence $[\![(\nu\widetilde{n})p]\!]_{L-\{\widetilde{n}\}} \cong^{\mathtt{tr}}_{cK} [\![(\nu\widetilde{n})q]\!]_{L-\{\widetilde{n}\}}$. By definition of $\mathfrak{R}$, this suffices to conclude.
- We want to prove that $p|r\ \mathfrak{R}\ q|r$ by knowing that $[\![p]\!]_L \parallel [\![r]\!]_{L'} \cong^{\mathtt{tr}}_{cK} [\![q]\!]_L \parallel [\![r]\!]_{L'}$. By definition of $[\![\cdot]\!]$ and by Proposition 6.2.2, it holds that $[\![\cdot]\!]_L \parallel$

$[\![r]\!]_{L'} \gtrsim_{cK} [\![\cdot]\!]_{L\cup L'} \parallel [\![r]\!]_{L\cup L'} \triangleq [\![\cdot \parallel r]\!]_{L\cup L'}$. Thus, $[\![p \parallel r]\!]_{L\cup L'} \cong^{\mathtt{tr}}_{cK} [\![q \parallel r]\!]_{L\cup L'}$ and $fn(p|r, q|r) = fn(p, q) \cup fn(r) \subseteq L \cup L'$. This suffices to conclude.

We are left with proving the converse, i.e. $p \approx_{\pi_a} q$ implies that $[\![p]\!]_L \cong^{\mathtt{tr}}_{cK} [\![q]\!]_L$. Let $\mathfrak{R} \triangleq \{([\![p]\!]_L\ ,\ [\![q]\!]_L) : p \approx_{\pi_a} q\}$; we prove that $\mathfrak{R}$ is a barbed congruence, up-to $\lesssim_{cK}$. For reduction closure, we let $[\![p]\!]_L \xrightarrow{\tau} N$; by Lemma 6.4.2 it holds that $p \xrightarrow{\tau} p'$ and $N \gtrsim_{cK} [\![p']\!]_L$. Then, $q \Rightarrow q'$ and $p' \cong_{\pi_a} q'$. By Lemma 6.4.1, we know that $[\![q]\!]_L \Rightarrow [\![q']\!]_L$; this suffices to conclude up-to $\lesssim_{cK}$. Barb preservation can be proved easily. By Proposition 2.8.2, $[\![p]\!]_L \downarrow a$ implies that $[\![p]\!]_L \xrightarrow{(\nu\widetilde{b})\ \langle b\rangle\ @\ a}$; by Lemma 6.3.4 (or .5) and by definition of barbs in π_a-calculus, this implies that $p \downarrow \overline{a}$. Then, $q \Downarrow \overline{a}$; by using Lemmata 6.3.1 (or .2) and 6.4.1, we obtain the desired $[\![q]\!]_L \Downarrow a$.

To conclude, we have to prove that, for every translated context $\mathcal{C}[\cdot]$, it holds that $\mathcal{C}[[\![p]\!]_L]\ \mathfrak{R}\ \mathcal{C}[[\![q]\!]_L]$. The key observation is that, by definition of translated context, it holds that $\mathcal{C}[\cdot] \equiv (\nu\widetilde{n})([\cdot] \parallel [\![r]\!]_{L'})$. Moreover, by hypothesis, we know that $(\nu\widetilde{n})(p|r) \cong_{\pi_a} (\nu\widetilde{n})(q|r)$. Hence,

$$
\begin{array}{lll}
\mathcal{C}[[\![\cdot]\!]_L] \equiv & (\nu\widetilde{n})([\![\cdot]\!]_L \parallel [\![r]\!]_{L'}) & \\
\gtrsim_{cK} & (\nu\widetilde{n})([\![\cdot]\!]_L \parallel [\![r]\!]_{L'}) & \text{by Prop. 6.2.3} \\
\gtrsim_{cK} & (\nu\widetilde{n})([\![\cdot]\!]_{L\cup L'} \parallel [\![r]\!]_{L\cup L'}) & \text{by Prop. 6.2.2} \\
\triangleq & (\nu\widetilde{n})(nrm_{L\cup L'}([\![\cdot \mid r]\!]_{L\cup L'})) & \\
\triangleq & nrm_{L''}((\nu\widetilde{n})[\![\cdot \mid r]\!]_{L\cup L'}) & \text{for } L'' \triangleq (L \cup L') - \{\widetilde{n}\} \\
\triangleq & nrm_{L''}((\nu\widetilde{n})(\mathtt{proc} :: [\![\cdot \mid r]\!] \ \parallel \prod_{l'\in L\cup L'} l' :: \mathbf{nil})) & \\
\equiv & nrm_{L''}((\nu\widetilde{n})(\mathtt{proc} :: [\![\cdot \mid r]\!])) \ \parallel \prod_{l'\in L''} l' :: \mathbf{nil} & \\
= & nrm_{L''}(\mathtt{proc} :: [\![(\nu\widetilde{n})(\cdot \mid r)]\!]) \ \parallel \prod_{l'\in L''} l' :: \mathbf{nil} & \text{by Prop. 6.2.1} \\
\equiv & [\![(\nu\widetilde{n})(\cdot \mid r)]\!]_{L''} &
\end{array}
$$

Notice that, if $fn(\cdot) \subseteq L$ and $fn(r) \subseteq L'$ (these hold by definition of the encoding), then $fn((\nu\widetilde{n})(\cdot|r)) \subseteq (L \cup L') - \{\widetilde{n}\} \triangleq L''$. Thus, $\mathcal{C}[[\![p]\!]_L] \gtrsim_{cK} [\![(\nu\widetilde{n})(p|r)]\!]_{L''}$ $\mathfrak{R}$ $[\![(\nu\widetilde{n})(q|r)]\!]_{L''} \lesssim_{cK} \mathcal{C}[[\![q]\!]_L]$. This suffices to conclude, up-to $\lesssim_{cK}$. □

Corollary 6.6 (Full Abstraction w.r.t. Translated Barbed Equivalence) *Let $fn(p, q) \subseteq L$. Then $p \cong_{\pi_a} q$ if and only if $[\![p]\!]_L \cong^{\mathtt{tr}}_{cK} [\![q]\!]_L$.*

Proof: Trivial, by Theorem 6.5, Proposition 6.2.3 and by observing that $\lesssim_{cK} \subseteq \cong_{cK} \subseteq \cong^{\mathtt{tr}}_{cK}$. □

Remark 6.7: On full abstraction w.r.t. barbed equivalence. We have already said that translated full abstraction seems us the best possible result for the encod-

ing of Table 14. Indeed, there is a key design issue that breaks full abstraction: in π-calculus, knowing a name implies that communication actions can be performed upon a channel with that name and these actions succeed whenever a parallel component performs a complementary action. This is not the case in KLAIM (and in the calculi derived from it). Indeed, it is *not* necessarily the case that each free name is associated to a locality (while each name in a π-calculus process is associated to a channel). This aspect can break full abstraction: e.g., consider the following π_a-calculus equivalence

$$p \triangleq a(x).(\overline{x} \mid x\overline{b}) \quad \cong_{\pi_a} \quad a(x).(\,(\overline{x} \mid x\overline{b}) \oplus \overline{b}\,) \triangleq q$$

where $\oplus$ denotes internal choice. However,

$$[\![p]\!]_L \quad \cong_{cK} \quad [\![q]\!]_L$$

does not hold. Indeed, $[\![q]\!]_L$ can produce a datum at node b, while $[\![p]\!]_L$ cannot: if the name received in the input (that replaces x) is not a node of the net, the encoding of the output over x will never produce a datum. Thus, the input from x is blocked and the following output on b will never produce a datum.

We think that no 'reasonable' encoding of π_a-calculus in cKLAIM (nor in any other calculus derived from KLAIM) can be given: checking the existence of nodes before firing an output is a too low-level feature that cannot be implemented in such an abstract setting as π-calculus. There are two ways in which we can recover full abstraction.

(1) We can make cKLAIM higher-level: a simple way to do this is to add the following structural rule to those given in Table 3

$$l :: \mathbf{nil} \equiv \mathbf{0}$$

In this way, we recover π-calculus' philosophy that each name is *always* associated to a communication medium (up-to $\equiv$).

Another possibility is to consider a typed language, where types ensure that, if a locality name is eventually used as target of an operation, then a node whose address is that name is present in the net. This possibility strongly resembles Dπ's framework [20].

(2) We can make π_a-calculus lower-level: some names are channels, while the other ones are just communicable objects. This can be formalized by structuring the syntax of π_a-calculus as follows:

$$\begin{array}{ll} \text{Systems} & S ::= \exists a \mid (\nu a)S \mid S_1 \mid S_2 \mid p \\ \text{Processes} & p ::= \ldots \end{array}$$

where the particle $\exists a$ implements the presence of a channel with name a. The operational semantics of Table 13 must be then modified by following the

Encoding Nets:

$$(\!|\mathbf{0}|\!) \triangleq \overline{\mathtt{ex}}\langle\rangle \qquad (\!|l :: C|\!) \triangleq (\!|C|\!)_l \mid !\,\overline{\mathtt{ex}}\,l$$

$$(\!|(\nu l)N|\!) \triangleq (\nu l)((\!|N|\!) \mid !\,\overline{\mathtt{ex}}\,l) \qquad (\!|N_1 \parallel N_2|\!) \triangleq (\!|N_1|\!) \mid (\!|N_2|\!)$$

Encoding Processes:

$$(\!|\mathbf{nil}|\!)_u \triangleq \mathbf{0} \qquad (\!|\langle l\rangle|\!)_u \triangleq \bar{u}\,l$$

$$(\!|X|\!)_u \triangleq X \qquad (\!|\mathbf{rec}\,X.P|\!)_u \triangleq \mathbf{rec}\,X.(\!|P|\!)_u$$

$$(\!|C_1|C_2|\!)_u \triangleq (\!|C_1|\!)_u \mid (\!|C_2|\!)_u \qquad (\!|\mathbf{new}(l).P|\!)_u \triangleq (\nu l)((\!|P|\!)_u \mid !\,\overline{\mathtt{ex}}\,l)$$

$$(\!|\mathbf{out}(u').P|\!)_u \triangleq \bar{u}\,\acute{u} \mid (\!|P|\!)_u \qquad (\!|\mathbf{in}(!x).P|\!)_u \triangleq u(x).(\!|P|\!)_u$$

$$(\!|\mathbf{in}(u').P|\!)_u \triangleq \mathbf{rec}\,X.u(x).(\nu c)\big(\ \bar{c} \mid [x = \acute{u}]c.(\!|P|\!)_u \mid c.(\bar{u}\,x \mid X)\big) \qquad (*)$$

$$(\!|\mathbf{eval}(Q)@u'.P|\!)_u \triangleq \mathbf{rec}\,X.\mathtt{ex}(x).(\nu c)\big(\ \bar{c} \mid [x = \acute{u}]c.((\!|P|\!)_u \mid (\!|Q|\!)_{u'}) \mid c.X\ \big) \qquad (*)$$

$(*)$ for x, X fresh

Table 15
Encoding lcKlaim in π_a-calculus

lines of the LTS in Table 7 (by adding a check of existence of a channel before firing an output action).

6.2 Encoding lcKlaim *in* π_a*-calculus*

We now present an encoding of the simplest Klaim-based calculus, namely lcKlaim, in π_a-calculus. The encoding is somehow inspired from the encoding of Klaim in μKlaim (for the handling of names) and of μKlaim in cKlaim (for the encoding of the name matching construct of lcKlaim).

We can follow the correspondence between channels and localities that we pointed out in Section 6.1 and translate each locality to a channel. Output actions performed at l, as well as data located at l, can be translated to output particles of π_a-calculus $\bar{l}\,\_$. Similarly, input actions performed at l can be translated to input prefixes of π_a-calculus $l(x).\,\_$. Finally, any action **new**(l') is translated to a restriction $(\nu l')$. Thus, the correspondence between the two calculi is quite straightforward up to now.

A first feature that distinguish lcKlaim from π_a-calculus is the communication paradigm and, mainly, the name matching of lcKlaim (that happens while retriev-

ing a datum). This issue can encoded quite easily, if we accept divergence: process **in**(l').P running at l can be translated into a process that first retrieves a datum at l and then checks if it is l'; if the check succeeds, the process continues, otherwise it places back the accessed datum and looks for another one.

A second feature that distinguish lcKlaim from π_a-calculus is the allocation of processes and their movements, together with the check of locality existence before migration. Process distribution is relevant in lcKlaim to establish where actions **out** and **in** have to take place. Thus, we can define a parameterized encoding for processes, $(\!|P|\!)_u$, where u is the locality where P runs. Then, if P is of the form **out**(u').Q, we translate it to $\overline{u}u' \mid (\!|Q|\!)_u$, while, if P is of the form **in**($!x$).Q, we translate it to $u(x).(\!|Q|\!)_u$. A process P of the form **eval**(Q)@u'.R running at u is translated to the parallel composition of $(\!|R|\!)_u$ and $(\!|Q|\!)_{u'}$, if existence of locality u' is ascertained.

The last feature we have to model is the distinction between names that are addresses of network nodes and raw names. The former ones can be then used as target of remote operations (in the case of lcKlaim only actions **eval**), while the latter ones cannot. By using a π-calculus terminology, only the first ones are *Names* (we intentionally used the capital letter), while the latter ones are just *values*. However, the status of a name (without the capital letter) can change according to the context: a Name will always remain such in any context, while a value l can become a Name if the context provides a node with address l.

To deal with this sophisticated feature (that, as we have already discussed in Remark 6.7, creates a relevant gap between π_a-calculus and Klaim-based calculi), we use a reserved channel $\mathtt{ex}$ to record existence of localities. Thus, if l is a Name in the lcKlaim net considered for translation, then channel $\mathtt{ex}$ will repeatedly offer l in the encoded net, i.e. the encoding will contain a process of the form

$$!\,\overline{\mathtt{ex}}\,l \;\triangleq\; (\nu c)(\overline{c}\,l \mid !c(x)(\overline{c}\,x \mid \overline{\mathtt{ex}}\,x))$$

The encoding is summarized in Table 15. There, we also assume the possibility of writing π_a-calculus processes with recursion – that can be implemented through replication, as usual – and we write $\overline{c}$ and c to mean output and input of dummy data. In the translation of actions **in**(l) and **eval**, the fresh restricted channel c is used to implement a form of *internal choice*. In both cases, the first addendum can evolve only if the name matching succeeds. On the other hand, the second addendum can always be executed: this fact introduces divergence in the encoding. Notice, however, that exactly one of the two addenda can evolve. Finally, like in the encoding of Klaim in μKlaim, the fact that $\mathtt{ex}$ always provides data is necessary to obtain a fully abstraction result w.r.t. translated contexts. Again, translated contexts do not have a full discriminating power over this channel.

The proof of soundness somehow follows proofs already given in the paper; we

only sketch the main steps and leave the details to the interested reader. First, the translated barbed expansion in π_a-calculus, written $\lesssim^{tr}_{\pi_a}$, can be defined by following Definition 2.11. By following Proposition 2.12, it can be proved that $\lesssim^{tr}_{\pi_a} \subset \cong^{tr}_{\pi_a}$ and, by following Lemma 2.15, that translated barbed congruence up-to $\lesssim^{tr}_{\pi_a}$ is contained in $\cong^{tr}_{\pi_a}$. Then, we can prove that each lcKlaim reduction is preserved by its encoding.

Lemma 6.8 *If $N \longmapsto N'$, then $(\!|N|\!) \Rightarrow \gtrsim^{tr}_{\pi_a} (\!|N'|\!)$.*

Now, since the encoding $(\!|\cdot|\!)$ is divergent, we can follow the ideas of Section 4 and define partial reducts and partial states. Notice that, since divergence can originate both from the encoding of name matching and of migrations, we have two possible cases for partial reducts.

Definition 6.9 *(1) A π_a-calculus process p is a* partial reduct *of a* lcKlaim *net N whenever*

- $N \equiv l :: \mathbf{in}(l').P \mid \langle l' \rangle$ *and* $p \cong^{tr}_{\pi_a} (\nu c)\big(\overline{c} \mid [x = l]c.(\!|P|\!)_u \mid c.(\overline{l}\, x \mid (\!|\mathbf{in}(T).P|\!)_l)\big) \mid !\,\overline{\mathtt{ex}}\, l$, *or*
- $N \equiv l :: \mathbf{eval}(Q)@l'.P \parallel l' :: \mathbf{nil}$ *and* $p \cong^{tr}_{\pi_a} (\nu c)\big(\overline{c} \mid [x = l]c.((\!|P|\!)_l \mid (\!|Q|\!)_{l'}) \mid c.(\!|\mathbf{eval}(Q)@l'.P|\!)_l\big) \mid !\,\overline{\mathtt{ex}}\, l \mid !\,\overline{\mathtt{ex}}\, l'$.

(2) A π_a-calculus process p is a partial state *of a* lcKlaim *net N whenever $N \equiv (\nu \widetilde{l})(N_1 \parallel \cdots \parallel N_n \parallel \overline{N})$, $p \equiv_\pi (\nu \widetilde{l})(p_1 \parallel \cdots \parallel p_n \parallel (\!|\overline{N}|\!))$ and for all indexes i it holds that p_i is a partial reduct of N_i (where $\equiv_\pi$ is Milner's structural equivalence, see [22]).*

The pleasant property of μKlaim's partial reducts is here stronger.

Lemma 6.10 *Let p be a partial reduct of N. Then,*

- *$N \equiv l :: \mathbf{in}(l').P \mid \langle l' \rangle$ and $p \xrightarrow{\tau} p'$ imply that either $p' \gtrsim^{tr}_{\pi_a} (\!|N|\!)$, or $p' \gtrsim^{tr}_{\pi_a} (\!|P|\!)_l \mid !\,\overline{\mathtt{ex}}\, l$.*
- *$N \equiv l :: \mathbf{eval}(Q).P \parallel l' :: \mathbf{nil}$ and $p \xrightarrow{\tau} p'$ imply that either $p' \gtrsim^{tr}_{\pi_a} (\!|N|\!)$, or $p' \gtrsim^{tr}_{\pi_a} (\!|P|\!)_l \mid (\!|Q|\!)_{l'} \mid !\,\overline{\mathtt{ex}}\, l \mid !\,\overline{\mathtt{ex}}\, l'$.*

We can now state the reflection of reduction steps.

Lemma 6.11 *If $\langle\!| N |\!\rangle \xrightarrow{\tau} p$, then either $N \xrightarrow{\tau} N'$ and $p \gtrsim^{tr}_{\pi_a} \langle\!| N' |\!\rangle$, or p is a partial state of N.*

Finally, it is easy to see that the encoding faithfully translates the barbs; it only adds new barbs on $\mathtt{ex}$ but no translated context can fully observe them. Hence, the proof of the following concluding theorem can be carried on easily.

Theorem 6.12 (Full Abstraction w.r.t. Translated Barbed Congruence) *$N \cong_{lcK} M$ if and only if $(\!|N|\!) \cong^{tr}_{\pi_a} (\!|M|\!)$.*

Proof: For the 'if' direction, it suffices to prove that relation

$$\begin{aligned}\mathfrak{R} \triangleq\ & \{(N, M) : (\!|N|\!) \cong^{\mathrm{tr}}_{\pi_a} (\!|M|\!)\} \\ & \cup \{(N, M) : \exists\, \overline{p}.\ (\!|N|\!) \stackrel{\mathrm{tr}}{\cong}_{\pi_a} \overline{p} \ \wedge\ \overline{p} \text{ partial state of } M\} \\ & \cup \{(N, M) : \exists\, \overline{p}.\ \langle M \rangle \stackrel{\mathrm{tr}}{\cong}_{\pi_a} \overline{p} \ \wedge\ \overline{p} \text{ partial state of } N\}\end{aligned}$$

is barb preserving, reduction closed (up-to $\lesssim_{lcK}$) and closed under translated contexts (again, up-to $\lesssim_{lcK}$). For the 'only if' direction, it suffices to prove that relation

$$\begin{aligned}\mathfrak{R} \triangleq\ & \bigcup_{N \cong_{lcK} M} \{(\, (\!|N|\!), (\!|M|\!)\,)\} \\ & \cup\ \{(\, (\!|N|\!), p) : p \text{ partial state of } M\} \\ & \cup\ \{(p, (\!|M|\!)) : p \text{ partial state of } N\}\end{aligned}$$

is barb preserving, reduction closed (up-to $\lesssim^{\mathrm{tr}}_{\pi_a}$) and closed under translated contexts (again, up-to $\lesssim^{\mathrm{tr}}_{\pi_a}$). □

7 Concluding Assessment

In this section we discuss the quality of the presented encodings, which should be measured by considering how faithful the target terms are to the source ones.

According to [26], each 'reasonable' encoding $enc(\cdot)$ should have a number of important features aiming at guaranteeing the same degree of parallelism, a close correspondence between the names of the used channels and the same semantics. In particular, an encoding has to:

(1) *be homomorphic w.r.t. the parallel operator,* i.e. $enc(N \parallel M) = enc(N) \parallel enc(M)$;
(2) *preserve renaming,* i.e. for every permutation of names σ in the source language there exists a permutation of names θ in the target language such that $enc(P\sigma) = (enc(P))\theta$;
(3) *preserve the basic observables*, i.e. it has to preserve the visible behaviours of the encoded terms;
(4) *preserve termination*, i.e. it has to turn terminating terms in terminating terms.

All the encodings presented in this paper enjoy properties 2. and 3. . Property 4. is not enjoyed by the encodings of μKLAIM in cKLAIM and of lcKLAIM in π_a-calculus. This is related to the fact that the kind of name matching used in KLAIM-based calculi (borrowed from LINDA [17]) is very powerful: it permits performing boolean tests on names while retrieving them. Property 1. (i.e. $[\![P \mid Q]\!] \equiv [\![P]\!] \mid [\![Q]\!]$) is not enjoyed by the encoding of π_a-calculus in cKLAIM and by the encoding of KLAIM

in μKLAIM. In this case, we needed to have a centralized entity (locality env) that coordinates the translation of names. According to [24], the presence of such centralized authorities does not necessarily imply that the encoding developed is weak – the resolution of names in the Internet (through the so called DNS) requires some form of centralized knowledge to turn logical names in IP addresses – and we believe that in this scenario property 1. could be relaxed. However, we have that homomorphic translations are guaranteed for nets (i.e. $[\![N \parallel M]\!] \equiv [\![N]\!] \parallel [\![M]\!]$).

Now consider the properties put forward in [26], the facts that $\cong^{\text{tr}}$ is coarser than $\cong$ and that semantical equivalence implies full abstraction (w.r.t. the same equivalence). By considering the results summarised in Table 1, we can order the different kinds of encodings obtained in this paper as follows:

$$\xrightarrow{\text{S.E. w.r.t. } \cong} \gg \xrightarrow{\text{F.A. w.r.t. } \cong^{\text{tr}}} \gg \dashrightarrow^{\text{F.A. w.r.t. } \cong^{\text{tr}}}$$

where '$\gg$' can be interpreted as 'better than'.

Thus, the encoding of cKLAIM in lcKLAIM is the best we can imagine: it does not introduce divergence and translates nets into barbed congruent ones. Hence, the two calculi have exactly the same expressive power; remote communications are just a mean to simplify programming.

The encodings of KLAIM in μKLAIM and of π_a-calculus in cKLAIM are satisfactory. Indeed, they enjoy the four properties of [26], the very same properties enjoyed by Milner's encoding of polyadic π-calculus into the monadic one [22]. The generated code (especially in the case of the second encoding) is quite simple. These consideration lead us to conclude that source and target languages of the two encodings have similar expressive power.

The encodings of μKLAIM in cKLAIM and of lcKLAIM in π_a-calculus are less satisfactory. The two encodings may introduce divergence, and the encoding of polyadic communication (μKLAIM) in monadic communication (cKLAIM) is neither simple nor efficient. Table 10 substantiates this claim: a lot of monadic exchanges are necessary to implement each polyadic communication. While this could be acceptable from a theoretical point of view, it is hardly usable in practice. Therefore, we conclude that, in a LINDA-like framework, these two forms of communication seem not interchangeable, and that μKLAIM appears to be more expressive than cKLAIM.

Clearly, the results in this paper do not prove that μKLAIM and lcKLAIM are more powerful than cKLAIM and π_a-calculus, respectively. To that aim, we should exhibit some impossibility results similar to that of [26]. We are currently working on proving the impossibility of encoding lcKLAIM into π_a-calculus. We conjecture that, due to the check of existence of the target of a communication, that is performed in lcKLAIM and not in π_a-calculus, this result should hold. About the encodability of a polyadic communication through monadic communications (μKLAIM into cKLAIM), we think that a divergence-free encoding does not exist. Indeed, the fields of

a polyadic datum can only be accessed sequentially and one is forced to split the atomic activity of function *match* into a field-by-field compliance checking. Such checking must be aborted as soon as the accessed datum does not match the template used to retrieve it, and the inputting process must be rolled back, to try with another datum. The possibility of repeatedly accessing the same (non-matching) datum clearly leads to divergence.

8 Conclusions

In this paper, we have presented a family of process description languages for network-aware programming. The starting point has been KLAIM, an experimental language combining the process algebra approach with the coordination-oriented one. We have distilled from KLAIM some more and more foundational calculi (namely, μKLAIM, cKLAIM and lcKLAIM) and have studied the encoding of each of them into a simpler one. The expressive power of KLAIM-based calculi has been finally tested by a comparison with asynchronous π-calculus. In the development of the last part, we choose a monadic version of π_a-calculus; this choice was only driven by the sake of simplicity. Indeed, the encoding presented in Section 6.1 can be readily accommodated to yield an encoding of polyadic π_a-calculus into μKLAIM.

In our view, the present work throws light on the expressiveness of KLAIM and vindicates the design choices that make it significantly different from standard process calculi. The results presented here can be exploited also for assessing expressiveness of other calculi with a similar communication paradigm. In particular, we intend to assess more deeply the expressive power of pattern-matching, by studying 'reasonable' encodings of calculi with communication based on pattern matching into calculi with simple channel-based communications. Moreover, this work also stimulated us to find other variants of KLAIM that better model more sophisticated settings. For example, in [13,15] we have extended μKLAIM with two typical features of global computers, namely *dynamic inter-node connections* and *failures*, while in [16] we have developed more flexible (but still easily implementable) forms of pattern matching. Of course, a lot of work remains to be done: e.g., the formal study of the expressive power of all these variants is still missing.

We have discussed throughout the paper the works on encodings of process calculi that are strictly related to ours. Here, we touch upon the impact on expressiveness of the three different semantics for the output operation studied in [7] in the setting of a simple Linda-based process calculus: *instantaneous output* (an output prefix immediately unleashes the corresponding tuple in the TS), *ordered output* (a reduction is needed to turn an output prefix into the corresponding tuple in the TS) and *unordered output* (two reductions are needed to turn an output into an available tuple, one sends the tuple to the TS and another makes the tuple available in the TS). According to this terminology, the semantics of KLAIM output operation is

ordered. In [7] it is proved that the instantaneous semantics yields the most expressive setting. We believe that the instantaneous semantics would simplify the theory developed in this paper. For example, the proofs for the encoding of π_a-calculus into cKLAIM would be simpler because any top-level action a target term intends to perform would correspond to an analogous action in the source term. However, instantaneous tuple emission is unrealistic, especially in a network-aware scenario where remote operations are possible. On the other hand, unordered outputs are very close to the practice of network-aware programming (consider, e.g., sending e-mail messages). We believe that the theory presented in this paper can be tailored to deal with such semantics. However, in [8] it is proved that the simple Linda-based process calculus considered in [7] is Turing powerful under the instantaneous and ordered semantics but not with the unordered semantics. The output operation of KLAIM represents a compromise between expressiveness and implementability.

Acknowledgements. We would like to thank Jos Baeten and Flavio Corradini for the invitation to present our results at the *EXPRESS'04* workshop; it was the main stimulus for this work. Many thanks also to Diletta Cacciagrano for suggestions and comments on an earlier draft. The anonynmous referees also helped in improving the overall presentation.

References

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

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

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

[4] L. Bettini and R. D. Nicola. Mobile distributed programming in x-klaim. In M. Bernardo and A. Bogliolo, editors, *SFM*, volume 3465 of *LNCS*, pages 29–68. Springer, 2005.

[5] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195(2):205–226, 1998.

[6] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[7] N. Busi, R. Gorrieri, and G. Zavattaro. Comparing three semantics for linda-like languages. *Theoretical Computer Science*, 240(1):49–90, 2000.

[8] N. Busi, R. Gorrieri, and G. Zavattaro. On the expressiveness of linda coordination primitives. *Information and Computation*, 156(1-2):90–121, 2000.

[9] D. Cacciagrano and F. Corradini. On synchronous and asynchronous communication paradigms. In *Proc. of ICTCS'01*, volume 2202 of *LNCS*, pages 256–268. Springer.

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

[11] 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.

[12] R. De Nicola, G. Ferrari, and R. Pugliese. Programming Access Control: The Klaim Experience. In *Proc. of CONCUR'00*, volume 1877 of *LNCS*, pages 48–65. Springer.

[13] R. De Nicola, D. Gorla, and R. Pugliese. Basic observables for a calculus for global computing. Tech. Rep. 07/2004, Dip. di Informatica, Università di Roma "La Sapienza". To appear in the *Proc. of ICALP'05*.

[14] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of KLAIM-based calculi. In *Proc. of EXPRESS'04*, ENTCS 128(2):117–130. Elsevier, 2004.

[15] 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. Springer, 2005.

[16] R. De Nicola, D. Gorla, and R. Pugliese. Pattern matching over a dynamic network of tuple spaces. In *Proc. of FMOODS'05*, volume 3535 of *LNCS*, pages 1–14. Springer.

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

[18] D. Gorla. Semantic Approaches to Global Computing Systems. *PhD Thesis*, Dip. Sistemi ed Informatica, Univ. di Firenze, 2005.

[19] D. Gorla and R. Pugliese. Resource access and mobility control with dynamic privileges acquisition. In *Proc. of ICALP'03*, volume 2719 of *LNCS*, pages 119–132. Springer, 2003.

[20] M. Hennessy and J. Riely. Resource Access Control in Systems of Mobile Agents. *Information and Computation*, 173:82–120, 2002.

[21] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. of ECOOP'91*, volume 512 of *LNCS*, pages 133–147. Springer, 1991.

[22] R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

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

[24] U. Nestmann. What is a 'good' encoding of guarded choice? *Information and Computation*, 156:287–319, 2000.

[25] U. Nestmann and B. C. Pierce. Decoding choice encodings. *Information and Computation*, 163:1–59, 2000.

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

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

[28] J. Riely and M. Hennessy. Trust and partial typing in open systems of mobile agents. In *Proc. of POPL '99*, pages 93–104. ACM, 1999.

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

[30] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001. To appear.

[31] N. Yoshida. Graph types for monadic mobile processes. In *Proc. of FST–TCS'96*, volume 1180 of *LNCS*, pages 371–386. Springer, 1996.