Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 286 (2012) 291–306

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Network Conscious π-calculus: A Concurrent Semantics

Ugo Montanari[1,2] Matteo Sammartino[1,3]

Computer Science Department
University of Pisa
Pisa, Italy

Abstract

Traditional process calculi usually abstract away from network details, modeling only communication over shared channels. They, however, seem inadequate to describe new network architectures, such as Software Defined Networks [1], where programs are allowed to manipulate the infrastructure. In this paper we present a network conscious, proper extension of the π-calculus: we add connector names and the primitives to handle them, and we provide a concurrent semantics. The extension to connector names is natural and seamless, since they are handled in full analogy with ordinary names. Our observations are multisets of routing paths through which sent and received data are transported. However, restricted connector names do not appear in the observations, which thus can possibly be as abstract as in the π-calculus. Finally, we show that bisimilarity is a congruence, and this property holds also for the concurrent version of the π-calculus.

Keywords: π-calculus, network-awareness, concurrent semantics

1 Introduction

The trend in networking is going towards more “open” architectures, where the infrastructure can be manipulated in software. This trend started in the nineties, when OpenSig [3] and Active Networks [24] were presented, but neither gained wide acceptance due to security and performance problems. More recently, OpenFlow [18,1] or, more broadly, Software Defined Networking has become the leading approach, supported by Google, Facebook, Microsoft and others. Software defined networks (SDNs) are networks in which a programmable controller machine manages a group of switches, by instructing them to install or uninstall forwarding rules and report traffic statistics.

[1] Research supported by the EU Integrated Project 257414 ASCENS

[2] e-mail: ugo@di.unipi.it

[3] e-mail: sammarti@di.unipi.it

doi:10.1016/j.entcs.2012.08.019

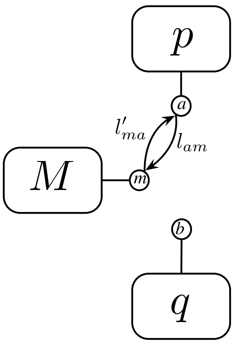
Traditional process calculi, such as π-calculus [21,22], CCS [20] and others, seem inadequate to describe these kinds of networks, because they abstract away from network details. In fact, two processes are allowed to communicate only through shared channels and it is not possible to express explicitly the fact that there is some complex connector between them. To give better visibility to the network architecture, in recent years network-aware extensions of known calculi have been devised [12,9].

This paper focuses on the π-calculus, and aims at equipping it with a natural notion of network: nodes and connectors are computational resources, so it is reasonable to represent them as (structured) names. We call the resulting calculus *Network Conscious π-calculus* (NCPi). We consider networks without hierarchies (e.g. administrative domains), where some parts may be private to a process and the public part is shared, as in CHARM [6]. Networks can be used by many processes at the same time, but we impose some restrictions on how resources can be accessed. The calculus has the following features:

- We distinguish two types of names: *sites*, which are the nodes of the network, and *links*, named connectors between pairs of sites. Sites are just atoms, e.g. a, links have the form l_{ab}, meaning that there is a link named l between a and b.
- The syntax can express the *creation* of a link through the restriction operator, and the *activation* of a transportation service over a link through a dedicated prefix. Separating these operations agrees with the π-calculus, where creating and using a channel as subject are two distinct operations. Moreover, since processes are not required to communicate on shared channels, an extended output primitive is introduced that specifies not only the emission site but also the destination one.
- We provide a concurrent semantics, where concurrent transmissions can be observed in the form of a multiset of routing paths. The associated behavioral equivalence is a *congruence*.

We choose to have labelled connectors, instead of anonymous ones as in [12] and [9], for two main reasons. First of all, they are intended to model transportation services with distinct features (cost, bandwidth. . .), which could be encoded in the label type, as we already do for the connectors' source and target. In any case, NCPi allows one to recover anonymous connectors through the restriction operator. Second, this enables reusing most of the notions of the π-calculus (renaming, α-conversion, extrusion. . .), suitably extended.

The main result of this paper is that bisimilarity on our concurrent semantics is a congruence. This is a desirable property for a process calculus, because it allows for the compositional analysis of systems. The authors of [9,12] treat bisimilarity and achieve compositionality as well, but they take a different approach than ours: they start from a reduction semantics, guess a suitable notion of barb, define barbed congruence by closing w.r.t. all the contexts, and then characterize it as a bisimulation equivalence on a labelled version of the transition system. In general, this approach yields labelled transition systems with succinct observations, but may resort to non-standard notions of bisimilarity, where the closure under contexts is "hardwired".



Processes

$$
\begin{aligned}
M &= m(x).m(y).(l_{xy})(\overline{m}xl_{xy}.M)\\
p &= \overline{a}ma.\overline{a}mb.a(l_{(xy)}).(L(l_{xy}) \,|\, \overline{a}bc.p')\\
q &= b(x).q'\\
L(l_{xy}) &= l_{xy}.L(l_{xy})
\end{aligned}
$$

System

$$S = p \,|\, M \,|\, q \,|\, L(l_{am}) \,|\, L(l'_{ma})$$

Fig. 1. Example system.

We show that we can gain the congruence property through a concurrent semantics, while keeping the notion of bisimilarity as standard as possible. We emphasize that interleaving semantics is far from being natural in this distributed setting. In fact, it is based on a mutual exclusion mechanism between remote actions which is simpler from a formal point of view, but not realistic for modeling concurrent systems.

Bisimilarity not being a congruence for the π-calculus depends on the interleaving nature of the semantics, and not on the language itself. In fact, we will show that, if we equip π-calculus with a concurrent semantics, the congruence property holds. This has already been shown in [16,17], but the semantics presented there allows observing the channel where a synchronization is performed, whereas our concurrent semantics is closer to the π-calculus, in the sense that we adopt a synchronization mechanism that hides such a channel.

The paper is organized as follows: in §2 we show a motivating example; in §3 we present the syntax of the language; in §4 we present the operational semantics and we show that bisimilarity is a congruence; in §5 we model a simple routing protocol. An extended version of this paper, including also an interleaving semantics, is available [23].

2 Motivating example

We consider the system shown in Fig. 1, made of a network manager M, using a reserved site m, and two processes p and q, which access the network respectively through the sites a and b. The manager is the only entity that can create new links and grant access to them. The process p wants to send a message to q, but we assume that there are no links between a and b allowing p and q to communicate.

The processes act as follows: M receives two sites at m, creates a new link between them and sends this link from m to the first of the received sites. The process p sends a and b from a to m, waits for a link at a and then evolves to the parallel composition of two components: the first component activates a transportation service over the received link, which can be used by the other component; the second component sends c from a to b. The process q simply waits for a datum at b. Finally, the process L, in order to simulate a persistent connection, repeatedly activates a transportation service over its argument: this is necessary, because the link prefix expresses a single activation of the service, as input/output prefixes in the π-calculus express a single usage of their subject channel.

We have that p, $L(l_{am})$ and M can do the following transitions

$$\bar{a}ma.\bar{a}mb.a(l_{(xy)}).(L(l_{xy}) \mid \bar{a}bc.p') \xrightarrow{\bullet;\bar{a}ma} \bar{a}mb.a(l_{(xy)}).(L(l_{xy}) \mid \bar{a}bc.p')$$

$$L(l_{am}) \xrightarrow{a;l_{am};m} L(l_{am})$$

$$m(x).m(y).(l_{xy})(\overline{m}xl_{xy}.M) \xrightarrow{mma;\bullet} m(y).(l_{ay})\overline{m}al_{ay}.M$$

where $\bullet;\bar{a}ma$ represents the beginning of transmission as a path of length zero, analogous to the π-calculus output action: the $\bullet$ on the left side indicates that the path can only extend rightward, i.e. subsequent hops will be listed after $\bullet$ from left to right in the form of a sequence of links; the string $\bar{a}ma$ describes the path, telling (from left to right) the site where the datum is available, the destination site and the datum. Symmetrically, $mma;\bullet$ means that a, which has destination m, is received at m and then goes through a path of length zero; it is analogous to the π-calculus input action. In this case the destination and reception site coincide, as in the input prefix ($a(x)$ can be thought of as $aa(x)$), but the reception site will eventually become different as the path grows. The label $a;l_{am};m$ represents the activation of a transportation service over l_{am}.

When these processes are put in parallel in S, their paths can be concatenated and a path representing a complete transmission on l_{am} can be observed

$$S \xrightarrow{\bullet;l_{am};\bullet} \bar{a}mb.a(l_{(xy)}).(L(l_{xy}) \mid \bar{a}bc.p') \mid m(y).(l_{ay})(\overline{m}al_{ay}.M) \mid q \mid L(l_{am}) \mid L(l'_{ma})\ .$$

As in the π-calculus, the transmitted datum, namely a, is not observable. Then, a sequence of possible transitions after this one is:

$$\cdots \xrightarrow{\bullet;l_{am};\bullet} a(l_{(xy)}).(L(l_{xy}) \mid \bar{a}bc.p') \mid (l_{ab})(\overline{m}al_{ab}.M) \mid q \mid L(l_{am}) \mid L(l'_{ma}) \qquad \text{(transmission of } b\text{)}$$

$$\xrightarrow{\bullet;l'_{ma};\bullet} (l_{ab})(L(l_{ab}) \mid \bar{a}bc.p' \mid M) \mid q \mid L(l_{am}) \mid L(l'_{ma}) \qquad (l_{ab} \text{ scope extension, } l_{ab} \notin \mathrm{fn}(p'))$$

$$\xrightarrow{\bullet;\bullet} (l_{ab})(L(l_{ab}) \mid p' \mid M) \mid q'[{}^{c}/_{x}] \mid L(l_{am}) \mid L(l'_{ma}) \qquad \text{(transmission of } c\text{)}$$

Notice that the last transition hides the link used for transmission, namely l_{ab}, because it is restricted. We just observe $\bullet;\bullet$, analogous to the π-calculus τ.

The semantics also allows observing in parallel all the pieces of a path. For instance, we can observe S doing $\bullet;\bar{a}ma \mid a;l_{am};m \mid mma;\bullet$, which represents a three-element multiset. These kinds of observations are exactly those making the behavioral equivalence compositional.

3 Syntax

We assume to have an enumerable set of site names $\mathcal{S}$ (or just sites) and an enumerable family of enumerable, disjoint sets of link names $\{\mathcal{L}_{a,b}\}_{a,b\in\mathcal{S}}$ (or just links). We let $\mathcal{L}_a = \biguplus_{b\in\mathcal{S}} \mathcal{L}_{a,b} \uplus \mathcal{L}_{b,a}$ and $\mathcal{L} = \biguplus_{a,b\in\mathcal{S}} \mathcal{L}_{a,b}$, and we denote by l_{ab} the element in $\mathcal{L}$ corresponding to $l \in \mathcal{L}_{a,b}$. Notice that, being $\mathcal{L}_{a,b}$ and $\mathcal{L}_{c,d}$ disjoint, for all $ab \neq cd$, we cannot have two links l_{ab} and l_{cd} unless $a = c$ and $b = d$.

Definition 3.1 NCPi processes are defined as follows, for $a, b \in \mathcal{S}, l_{ab} \in \mathcal{L}$:

$$
\begin{array}{l}
p ::= \mathbf{0} \mid \pi.p \mid p+p \mid p|p \mid (r)p \mid A(r_1, r_2, \ldots, r_n) \\
r ::= a \mid l_{ab} \qquad s ::= a \mid l_{(ab)} \qquad \pi ::= \overline{a}br \mid a(s) \mid l_{ab} \mid \tau \\
\quad A(s_1, s_2, \ldots, s_n) \overset{\text{def}}{=} p \qquad i \neq j \Rightarrow \mathrm{n}(s_i) \cap \mathrm{n}(s_j) = \varnothing
\end{array}
$$

Here we write $\mathrm{n}(s)$ for the names in s, including a and b if s is $l_{(ab)}$ (analogously for $\mathrm{n}(l_{ab})$). We have the usual inert process, sum and parallel composition. For the recursive definition, we require that formal parameters do not have names in common, because otherwise we might have type dependencies between parameters, e.g. in $A(a, l_{(ab)})$ one of the second parameter's endpoints depends on the first parameter. Prefixes can have the following forms:

- The *output prefix* $\overline{a}br$: $\overline{a}br.p$ can send the datum r from a addressed to b and continue as p. Notice that, unlike π-calculus, the destination site can be different than the emission one.
- The *input prefix* $a(s)$: $a(s).p$ can receive at a a datum to be bound to s and continue as p. The intended meaning of $c(l_{(ab)}).p$ is an atomic, polyadic version of $c(a).c(b).c(l_{ab}).p$. Here a monadic link input prefix $c(l_{ab}).p$ is not allowed, since it would introduce a matching capability we prefer not to provide. Consequently, a and b are not free in $c(l_{(ab)}).p$.
- The τ *prefix*: $\tau.p$ can perform an internal action and continue as p.
- The *link prefix* l_{ab}: $l_{ab}.p$ can offer to the environment the service of transporting a datum from a to b through l and then continue as p.

Finally, we have the *restriction* (r): r is private in $(r)p$, i.e. it cannot be observed as free name in a communication. Notice that a and b are free in $(l_{ab})p$. Sequences of restrictions will be denoted by capital letters (R).

We define the set $\mathrm{fn}(p)$ of free names of p as:

$$
\begin{aligned}
\mathrm{fn}(\mathbf{0}) &= \varnothing \\
\mathrm{fn}(\overline{a}br.p) &= \{a, b\} \cup \mathrm{n}(r) \cup \mathrm{fn}(p) \\
\mathrm{fn}(b(a).p) &= \{b\} \cup (\mathrm{fn}(p) \setminus (\{a\} \cup \mathcal{L}_a)) \\
\mathrm{fn}((a)p) &= \mathrm{fn}(p) \setminus (\{a\} \cup \mathcal{L}_a) \\
\mathrm{fn}((l_{ab})p) &= \{a, b\} \cup \mathrm{fn}(p) \setminus \{l_{ab}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{fn}(\tau.p) &= \mathrm{fn}(p) \\
\mathrm{fn}(l_{ab}.p) &= \{l_{ab}, a, b\} \cup \mathrm{fn}(p) \\
\mathrm{fn}(a(l_{(bc)}).p) &= \{a\} \cup \mathrm{fn}(p) \setminus \\
&\qquad (\{b, c\} \cup \mathcal{L}_b \cup \mathcal{L}_c) \\
\mathrm{fn}(p+q) = \mathrm{fn}(p \,|\, q) &= \mathrm{fn}(p) \cup \mathrm{fn}(q) \\
\mathrm{fn}(A(r_1, \ldots, r_n)) &= \mathrm{n}(r_1) \cup \cdots \cup \mathrm{n}(r_n)
\end{aligned}
$$

where $A(s_1, \dots, s_n) \stackrel{\text{def}}{=} p$ implies $\text{fn}(p) \subseteq \text{n}(s_1) \cup \dots \cup \text{n}(s_n)$. Notice the definition of $\text{fn}((a)p)$: if a link having a as one of its endpoints appears in p, then it is considered bound. Similarly for $b(a).p$ and $a(l_{(bc)}).p$. This intuitively means that a global link cannot have private endpoints, analogously to what happens for free processes in a well-formed state of a CHARM [6]: their variables must belong to the global part.

The notion of renaming is defined as follows.

Definition 3.2 A renaming σ is a pair of functions $\langle \sigma_{\mathcal{S}} : \mathcal{S} \to \mathcal{S}, \sigma_{\mathcal{L}} : \mathcal{L} \to \mathcal{L} \rangle$ such that $\sigma_{\mathcal{L}}(l_{ab}) = l'_{a'b'}$ implies $\sigma_{\mathcal{S}}(a) = a'$ and $\sigma_{\mathcal{S}}(b) = b'$. We denote by $r\sigma$ the result of applying the appropriate components of σ to r.

The condition relating $\sigma_{\mathcal{S}}$ and $\sigma_{\mathcal{L}}$ ensures that σ acts as a graph homomorphism, i.e. each link is renamed by $\sigma_{\mathcal{L}}$ to a link whose endpoints are the image through $\sigma_{\mathcal{S}}$ of the original link's endpoints. Some notation: we write $[{}^{r'_1}\!/r_1, {}^{r'_2}\!/r_2, \dots, {}^{r'_n}\!/r_n]$ to indicate the function mapping r_1 to r'_1, r_2 to r'_2 … r_n to r'_n, and we write $[{}^{l'_{a'b'}}\!/l_{(ab)}]$ as a shorthand for $[{}^{a'}\!/a, {}^{b'}\!/b, {}^{l'_{a'b'}}\!/l_{ab}]$. Notice that $[{}^{a'}\!/a]$ does not uniquely characterize a renaming. In fact, while certainly $\overline{a}bc[{}^{a'}\!/a] = \overline{a'}bc$, $a \notin \{b, c\}$, for $l_{ab}[{}^{a'}\!/a]$ we only know that it must belong to $\mathcal{L}_{a'b}$. Thus we should avoid applying such renaming to a link l_{ab}, since the result would be undefined. A special case (see below) is when l_{ab} is bound.

Now we introduce *well-formed* NCPi processes. Informally, a process is well-formed if each bound link it contains is bound explicitly, and not as a side-effect of binding a site, and if two links with the same label but different endpoints do not appear free in any of its suprocesses. For instance, $a(b).l_{bc}.p$ and $(l_{ab})l_{ab}.l_{cd}.p$ are not well-formed: the former because l_{bc} is implicitly bound by $a(b)$, the latter because l labels two links between different sites.

Definition 3.3 A NCPi process p is *well-formed* if for every subterm q:

(i) $q = (a)p'$ implies $\text{fn}(q) = \text{fn}(p') \setminus \{a\}$;

(ii) $q = b(a).p'$ implies $\text{fn}(q) = \{b\} \cup \text{fn}(p') \setminus \{a\}$;

(iii) $q = c(l_{(ab)}).p'$ implies $\text{fn}(q) = \{c\} \cup \text{fn}(p') \setminus \{a, b, l_{ab}\}$;

(iv) $l_{ab}, l'_{cd} \in \text{fn}(q)$ and $ab \neq cd$ implies $l \neq l'$.

A first consequence of this definition is that we do not need to subtract $\mathcal{L}_a$ or $\mathcal{L}_b \cup \mathcal{L}_c$ when computing the free names of $b(a).p$, $(a)p$ or $a(l_{(bc)})).p$, if these processes are well-formed.

Well-formedness also allows us to say how a generic substitution can act on processes as a proper renaming. This is needed in order to define α-conversion, which in fact is given in Fig. 2 for well-formed processes only. α-conversion for a restricted process is simply $(a)p \equiv (a')p[{}^{a'}\!/a]$, with $a' \notin \text{fn}((a)p)$, where $[{}^{a'}\!/a]$ is never applied to a link l_{ab}, since such link cannot be free in p. If it is bound, i.e. if $(l_{ab})p'$ is a subprocess of p, then we simply have inductively $((l_{ab})p')[{}^{a'}\!/a] \equiv (l'_{a'b})p'[{}^{l'_{a'b}}\!/l_{ab}][{}^{a'}\!/a]$, for any $l'_{a'b}$ such that $l'_{a''b''} \notin \text{fn}(p)$, for all a'', b''. Notice that, in order to mantain property (iv) of well-formedness, captures must be avoided not only in the presence of $l'_{a'b} \in \text{fn}(p)$, but also of links of the form $l'_{a''b''} \in \text{fn}(p)$, for any a'', b''. A similar restriction also holds when α-converting $a(l_{(bc)}).p$. We remark that these processes

α-equivalence:

$$(a)p \equiv (a')p[{}^{a'}/_{a}] \qquad b(a).p \equiv b(a').p[{}^{a'}/_{a}] \qquad a' \notin \mathrm{fn}((a)p)$$

$$\begin{array}{ll}
(l_{ab})p \equiv (l'_{ab})p[{}^{l'_{ab}}/_{l_{ab}}] & \forall a', b' : l'_{a'b'} \notin \mathrm{fn}((l_{ab})p) \\
a(l_{(bc)}).p \equiv a(l'_{(b'c')}).p[{}^{l'_{b'c'}}/_{l_{(bc)}}] & b', c' \notin \mathrm{fn}(a(l_{(bc)}).p) \wedge \\
 & \forall b'', c'' : l'_{b''c''} \notin \mathrm{fn}(a(l_{(bc)}).p)
\end{array}$$

Unfolding law: $A(r_1, \ldots, r_n) \equiv p[{}^{r_1}/_{s_1}, \ldots, {}^{r_n}/_{s_n}]$ if $A(s_1, \ldots, s_n) \overset{\mathrm{def}}{=} p$

Fig. 2. Structural congruence of well-formed processes.

can be α-converted also with respect to b,c or l_{bc} separately. In the following we will consider only well-formed processes.

4 Concurrent semantics

Interleaving semantics can be considered inadequate for distributed system with partially asynchronous behavior, since it implicitly assumes the existence of a central arbiter who grants access to resources. This criticism is particularly relevant for our network-conscious calculus. Here we present a concurrent semantics where we can observe multisets of routing paths covered at the same time. Single paths are denoted by α, multisets of paths by Λ and are called *concurrent paths.*

Definition 4.1 Paths and concurrent paths are defined as follows, for $a, b \in \mathcal{S}$, $l_{ab} \in \mathcal{L}$:

$$\begin{array}{l}
\alpha ::= a;W;b \mid \bullet;W;\bullet \mid \bullet;W;\overline{a}br \mid abr;W;\bullet \\
\quad \mid ab(s);W;\bullet \qquad \mathrm{n}(s) \cap (\mathrm{n}(W) \cup \{a, b\}) = \varnothing
\end{array}$$

$$r ::= a \mid l_{ab} \qquad s ::= a \mid l_{(ab)} \qquad W ::= l_{ab} \mid W;W \mid \epsilon$$

$$\Lambda ::= \mathbf{1} \mid \alpha \mid \Lambda_1|\Lambda_2 \mid (r)\Lambda$$

Their structural congruence $\equiv_\Lambda$ includes monoidality of ";", with ϵ as identity, and of $|$, with $\mathbf{1}$ as identity, and the following scope extension axioms:

$$\begin{array}{ll}
(r)(r')\Lambda \equiv_\Lambda (r')(r)\Lambda & r \notin \mathrm{n}(r'), r' \notin \mathrm{n}(r) \\
\Lambda_1 \,|\, (r)\Lambda_2 \equiv_\Lambda (r)(\Lambda_1 \,|\, \Lambda_2) & r \notin \mathrm{fn}(\Lambda_1)
\end{array}$$

A path α can be of two general forms. It can be a *service path* $a;W;b$, representing a transportation service from a to b that employs the resources listed in W and possibly other private, unobservable resources. Alternatively, it can be a sequence starting and/or ending with $\bullet$, which represents an actual transmission over W. More specifically, in this case α can be:

path α	fn	bn	obj	$\mathrm{obj}_{\mathrm{in}}$	is
$a; W; b$	$\mathrm{n}(\alpha)$	$\varnothing$	$\varnothing$	$\varnothing$	$\{a, b\}$
$\bullet; W; \bullet$	$\mathrm{n}(\alpha)$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\bullet; W; \overline{a}br$	$\mathrm{n}(\alpha)$	$\varnothing$	$\{b, r\}$	$\varnothing$	$\{a\}$
$abr; W; \bullet$	$\mathrm{n}(\alpha)$	$\varnothing$	$\{b, r\}$	$\{r\}$	$\{a\}$
$ab(s); W; \bullet$	$\mathrm{n}(\alpha) \setminus \mathrm{n}(s)$	$\mathrm{n}(s)$	$\{b\}$	$\varnothing$	$\{a\}$

Table 1
Free names, bound names, objects, input objects and interaction sites of a path α.

- an *output path*, if $\overline{a}br$ is on the right, representing the emission of r, whose destination is b, at a;
- an *input path*, if abr or $ab(s)$ is on the left. In the former case, it is called *free input path* and means that r, whose destination is b, is received at a; in the latter case, it is called *bound input path* and s is a placeholder for the received name;
- a *complete path*, if $\bullet$ is on both sides, meaning that the transmission has already been completed.

Concurrent paths can have the following forms:

- the *empty concurrent path* **1** indicates that no activity is performed;
- the *singleton concurrent path* α is a concurrent path made of a single path;
- the *union* $\Lambda_1 \mid \Lambda_2$ means that the paths in Λ_1 and Λ_2 are being traversed *at the same time*;
- the *extrusion restriction* $(r)\Lambda$ indicates that r is being extruded through one or more paths in Λ.

We will use W_α to denote the sequence of links of α and $|W_\alpha|$ to denote the set of links appearing in W_α. We call *interaction sites* of α, written is(α) and defined in Table 1, those sites where the interaction with another process may happen. These correspond to subjects of the π-calculus. Table 1 also defines the free names fn(), bound names bn(), objects obj(), input objects $\mathrm{obj}_{\mathrm{in}}()$. Their extensions to multisets is as expected. We have to be careful with the following cases:

$$\mathrm{fn}((a)\Lambda) = \mathrm{fn}(\Lambda) \setminus (\{a\} \cup \mathcal{L}_a) \qquad \mathrm{bn}((a)\Lambda) = \mathrm{bn}(\Lambda) \cup \{a\} \cup (\mathcal{L}_a \cap \mathrm{n}(\Lambda))$$

Notice that both the datum and the destination site are objects: this is analogous to actual routing, where a payload and its destination address travel together within a packet. We introduce some terminology for concurrent paths.

Definition 4.2 Let Λ be a concurrent path. Then it is:

- *well-formed* if for every subterm Λ' of the form $(a)\Lambda''$ we have $\mathrm{fn}(\Lambda') = \mathrm{fn}(\Lambda'') \setminus \{a\}$, and for all $l_{ab}, l'_{a'b'} \in \mathrm{fn}(\Lambda)$ we have $l \neq l'$ if $ab \neq a'b'$;
- in *canonical form* if it has the form $(R)\Theta$, where R is a sequence of restrictions and Θ does not contain extrusion restrictions (binders of the form $ab(s)$ are still allowed in Θ);
- *simple* if, for all $\alpha \in \Lambda$, each $l_{ab} \in |W_\alpha|$ appears in W_α once.

An example of non-well-formed concurrent path is $(d)(\bullet; l_{ab}; \overline{b}cd \mid a; l_{ad}; d)$, because (d) implicitly binds l_{ad}, and l_{ab}, l_{ad} have the same label but different endpoints. An example of non-simple concurrent path is $a; l_{ab}; l'_{cd}; l_{ab}; b$, because there are two occurrences of l_{ab}. Simplicity is just one of the possible conditions. In general, one might want to express QoS requirements: this could be achieved through a type system that associates quantitative information to links.

Now we introduce the *hiding operation*, which we will use in the SOS rules to implement the effects that restricting a name of a process has on its paths.

Definition 4.3 The hiding operation / acts on sequences of links as follows:

$$\epsilon/r = \epsilon \qquad (W;W')/r = (W/r);(W'/r) \qquad l_{ab}/r = \begin{cases} \epsilon & r \in \{a, b, l_{ab}\} \\ l_{ab} & \text{otherwise} \end{cases}$$

Its extension to paths α/r is obtained by replacing W_α with W_α/r in α. Its extension to concurrent paths Λ/r applies the same operation to each $\alpha \in \Lambda$ if $r \notin \mathrm{bn}(\Lambda)$, yields Λ otherwise.

We can now present the *NCPi transition system.*

Definition 4.4 The NCPi transition system is the smallest transition system generated by the rules in Fig. 3, where observations are up to $\equiv_\Lambda$ and transitions are closed under $\equiv$, i.e. if $p \xrightarrow{\Lambda} q$, $p \equiv p'$ and $q \equiv q'$ then $p' \xrightarrow{\Lambda} q'$.

The rules (FR-IN) and (BND-IN) infer the reception of a global and a private name, respectively, while (OUT) infers the emission of a global name. These actions are represented as paths of length zero. As in the early π-calculus, a renaming must be applied to the continuation in the free input case; by well-formedness, such renaming can always be extended to act as a proper graph homomorphism. The reception of a global link should be treated carefully: the rule forbids it whenever another link with the same label, but different endpoints, already occurs free in the process, because the renaming would break well-formedness. The rule (LINK) is used to provide a transportation service to the environment, but we forbid services from a site to itself. The rule (INTERNAL) infers a transition labelled with the empty path $\bullet;\bullet$, representing an internal action. The rule (RES) infers a transition of $(r)p$ from the transitions of p, but it considers only those transitions such that r is not an interaction site and is not sent or received. This side condition reflects that of the corresponding π-calculus rule, where r must not be the subject or the object of the premise's action, and its purpose is to avoid captures: e.g. if $(a)b(c).p$ is such that $c \in \mathrm{fn}(p)$ and it is allowed to perform $bba;\bullet$, then a would be captured in the continuation $(a)p[{}^a/_c]$. The rule (OPEN) infers a scope extrusion, provided that the name to extrude is used as object in the premise's concurrent path, but not as datum of an input or as interaction site. Notice that the rule allows one to "extrude" the destination site: the intuition is that we can use global resources to send or receive a datum to/from a local site, which becomes global if the communication is not complete. The rule (SUM-L) is an obvious extension of the corresponding π-calculus rule. The rule (IDLE) infers a "no-op" transition, enabling the parallel

(FR-IN)

$$\frac{}{a(s).p \xrightarrow{aar;\bullet} p[{}^{r}\!/_{s}]}\ r{=}l_{ab}{\Rightarrow}\forall a'b'{:}l_{a'b'}{\notin}\mathrm{fn}(a(s).p)$$

(BND-IN)

$$\frac{}{a(s).p \xrightarrow{aa(s);\bullet} p}$$

(OUT)

$$\frac{}{\overline{a}br.p \xrightarrow{\bullet;\overline{a}br} p}$$

(LINK)

$$\frac{}{l_{ab}.p \xrightarrow{a;l_{ab};b} p}\ a{\neq}b$$

(INTERNAL)

$$\frac{}{\tau.p \xrightarrow{\bullet;\bullet} p}$$

(RES)

$$\frac{p \xrightarrow{\Lambda} q}{(r)p \xrightarrow{\Lambda/r} (r)q}\ r{\notin}\mathrm{obj}(\Lambda){\cup}\mathrm{bn}(\Lambda){\cup}\mathrm{is}(\Lambda)$$

(OPEN)

$$\frac{p \xrightarrow{\Lambda} q}{(r)p \xrightarrow{(r)(\Lambda/r)} q}\ r{\in}\mathrm{obj}(\Lambda)\backslash(\mathrm{is}(\Lambda){\cup}\mathrm{obj}_{\mathrm{in}}(\Lambda))$$

(IDLE)

$$\frac{}{p \xrightarrow{\mathbf{1}} p}$$

(PAR)

$$\frac{p_1 \xrightarrow{\Lambda_1} q_1 \quad p_2 \xrightarrow{\Lambda_2} q_2}{p_1 \mid p_2 \xrightarrow{\Lambda_1 \mid \Lambda_2} q_1 \mid q_2}\quad \begin{array}{l} l_{ab}{\in}\mathrm{bn}(\Lambda_i){\cup}\mathrm{obj}_{\mathrm{in}}(\Lambda_i){\Rightarrow} \\ \quad \forall a'b'{\neq}ab{:}l_{a'b'}{\notin}\mathrm{n}(\Lambda_{3-i}){\cup}\mathrm{fn}(p_{3-i}),i{=}1,2 \\ \mathrm{bn}(\Lambda_i){\cap}(\mathrm{n}(\Lambda_{3-i}){\cup}\mathrm{fn}(p_{3-i})){=}\varnothing,i{=}1,2 \end{array}$$

(COM)

$$\frac{p \xrightarrow{(R)\ (\bullet;W;\overline{a}br \mid ab'x;W';\bullet \mid \Theta)} q}{p \xrightarrow{(R')\ (\bullet;W;W';\bullet \mid \Theta)} (R'')\ q(\sigma_b \circ \sigma_r)}\quad \begin{array}{l} R'{=}R{\cap}\mathrm{obj}(\Theta) \\ R''{=}(R\backslash R'){\cap}(\{b\}{\cup}\mathrm{n}(r)) \\ \text{see tables (b) and (c)} \end{array}$$

(SRV-IN)

$$\frac{p \xrightarrow{(R)\ (a;W;b \mid bcx;W';\bullet \mid \Theta)} q}{p \xrightarrow{(R)\ (acx;W;W';\bullet \mid \Theta)} q}$$

(SRV-OUT)

$$\frac{p \xrightarrow{(R)\ (\bullet;W;\overline{a}br \mid a;W';c \mid \Theta)} q}{p \xrightarrow{(R)\ (\bullet;W;W';\overline{c}br \mid \Theta)} q}$$

(SRV-SRV)

$$\frac{p \xrightarrow{a;W;b \mid b;W';c \mid \Lambda} q}{p \xrightarrow{a;W;W';c \mid \Lambda} q}$$

The concurrent path inferred by (COM), (SRV-IN), (SRV-OUT) and (SRV-SRV) must be simple.

(a)

• $b' = b$ • $\sigma_b = id$	• $b, b' \in R$ • $\sigma_b = [{}^{b}\!/_{b'}]$

(b)

• $r \notin R$ • $x = r$ • $\sigma_r = id$	• $r \in R$ • $x = (s)$ • $\sigma_r = [{}^{r}\!/_{s}]$

(c)

Fig. 3. NCPi operational rules: (a) shows the SOS rules; (b) and (c) are the possible configurations for (COM). Any pair of configurations, one from (b) and one from (c), is valid (four possibilities).

composition of processes to behave in an interleaving style. The rule (PAR) makes the union of two concurrent paths, but only if we do not lose well-formedness due to inconsistent link labels and if the concurrent path of each of its premise has bound names which are fresh w.r.t the other process and distinct from all the names occurring in the other concurrent path. This last condition avoids inferring transitions where the extruded name is free in the receiving process's continuation even if it has not been actually received, which might cause incorrect behaviors. For instance, consider the processes $p = (b)\overline{a}ab.b(c).p'$ and $q = a(d).\overline{d}de.q'$, and suppose $p \,|\, q \xrightarrow{(b)\bullet;\overline{a}ab \,|\, aab;\bullet} b(c).p' \,|\, \overline{b}be.q'[{}^{b}\!/\!{}_{d}]$ is allowed; now the two components of the continuation can synchronize on b even if its scope extension has not actually been accomplished, which is clearly incorrect.

The remaining rules are used to synchronize processes. The synchronization is performed in two steps: 1) paths of parallel processes are collected through the rule (PAR); 2) (COM), (SRV-IN), (SRV-OUT) and (SRV-SRV) take two compatible paths out of the resulting multiset and replace them with their concatenation, without modifying the source process; in other words, these rules synchronize two subprocesses of the source process. The rule (COM) covers all kinds of communications, yielding a complete path: the continuation is suitably renamed and, if the involved paths extrude some names, their restrictions are removed from the transition's label and added to the continuation, but only if there are no other paths extruding them. The rules (SRV-IN), (SRV-OUT) and (SRV-SRV) allow extending a path with a service path. The premises of (COM), (SRV-IN) and (SRV-OUT) must have their concurrent paths in canonical form: this is always possible, thanks to (PAR) side conditions.

The following proposition states that the transition system generated by these rules is well-behaved.

Proposition 4.5 *If $p \xrightarrow{\Lambda} q$ then Λ is simple and well-formed, and q is well-formed.*

We introduce the behavioral equivalence for NCPi processes, called *network conscious bisimilarity.*

Definition 4.6 A binary, symmetric and reflexive relation $\mathcal{R}$ is a *network conscious bisimulation* if $(p, q) \in \mathcal{R}$ and $p \xrightarrow{\Lambda} p'$, with:

(i) $\mathrm{bn}(\Lambda) \cap \mathrm{fn}(q) = \varnothing$;
(ii) $l_{ab} \in \mathrm{bn}(\Lambda) \cup \mathrm{obj}_{\mathrm{in}}(\Lambda) \Rightarrow \forall a'b' \neq ab : l_{a'b'} \notin \mathrm{fn}(q)$

implies that there is q' such that $q \xrightarrow{\Lambda} q'$ and $(p', q') \in \mathcal{R}$. The bisimilarity is the largest such relation and is denoted by $\sim^{NC}$.

Condition (i) is standard, while (ii) rules out the transitions of p that q may not be able to simulate due to well-formedness. Notice that a consequence of defining the semantics up to structural congruence is that $\equiv \subseteq \sim^{NC}$.

Theorem 4.7 *$\sim^{NC}$ is a congruence with respect to all NCPi operators.*

Proof. [Hint] This is proved by considering each possible elementary context and defining a suitable bisimulation closed under that context. The difficult case is the input prefix, since a renaming, possibly not injective, is involved. The key idea is that renaming a process may allow to apply more (COM), (SRV-IN), (SRV-OUT) or (SRV-SRV) rules, but the paths these rules concatenate are already observable in the original process, so the new transitions only depend on the original process' ones.□

We can establish a relation between a subcalculus of the interleaving NCPi and the π-calculus.

Proposition 4.8 *Let* linkless NCPi *be the subcalculus of NCPi such that no links appear in processes and the output prefix is of the form $\overline{a}ab$. Then there is a one-to-one correspondence:*

(i) *between π-calculus processes and NCPi processes;*

(ii) *between π-calculus transitions and NCPi transitions with singleton labels.*

This encoding maps $\overline{a}b$ to $\overline{a}ab$ or $\bullet;\overline{a}ab$, depending on whether it is used as prefix or as action; the other cases are obvious. By homomorphic extension we get the encoding for processes and transitions. Notice that (ii) does not rule out transitions with non-singleton concurrent paths occurring at intermediate derivation steps, e.g. those inferred by (PAR).

If we remove the restriction of having only singleton labels, we get a *concurrent π-calculus transition system.*

Corollary 4.9 (of Theorem 4.7) *The bisimilarity on the concurrent π-calculus transition system is a congruence.*

An evidence of this result is the classical counterexample not applying: we have $\overline{a}ar \,|\, a(x) \not\sim^{NC} \overline{a}ar.a(x) + a(x).\overline{a}ar$, because

$$\overline{a}ar \,|\, a(x) \xrightarrow{\bullet;\overline{a}ar \,|\, aar;\bullet} \mathbf{0} \qquad \overline{a}ar.a(x) + a(x).\overline{a}ar \not\xrightarrow{\bullet;\overline{a}ar \,|\, aar;\bullet}$$

This result is analogous to that in [17] but, as already mentioned, there the synchronization mechanism is not faithful to the π-calculus: in [17] the synchronization channel is observed unless restricted, for instance $\overline{a} \,|\, a \xrightarrow{\tau_a} \mathbf{0}$, while for our calculus $\overline{a} \,|\, a \xrightarrow{\bullet;\bullet} \mathbf{0}$, which corresponds to τ.

5 Example: routing protocol

In this section we model a simple routing protocol, similar to BGP [25]. This protocol assumes that the network is composed of disjoint groups of networks, each referring to a single administrative authority, called *Autonomous Systems* (AS). Some of the ASs' routers act as *gateways* between the AS they belong to and other networks. The protocol takes care of the routing mechanism between ASs in a distributed manner: each gateway has a *routing table*, filled by the protocol, whose entries specify which is the next hop along the "best" path towards some destination; this information will be used to forward the incoming data.

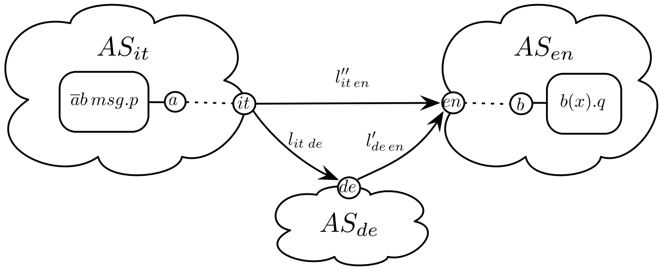


Fig. 4. Example network.

In our model, both routers and hosts are represented as sites, and network connections are represented as links. Autonomous systems are generic processes whose links are all restricted, because these links represent local services. The forwarding behavior of each gateway g is modeled as a process of the form $L(l^{(1)}_{gh_1}) \mid \cdots \mid L(l^{(n)}_{gh_n})$, where $L(l_{(xy)}) \overset{\text{def}}{=} l_{xy}.L(l_{xy})$, providing transportation services from g to other gateways. Routing tables are modeled as functions RT_g such that $RT_g(a)$ is a link l_{gh} to some other gateway h, representing the next hop of the best path towards a. The forwarding is implemented at the SOS level by employing the following rule for gateways [4]

$$(\text{FORWARD})\ \frac{p \xrightarrow{(R)\ (\bullet;W;\overline{g}ar\,|\,g;l_{gh};h\,|\,\Theta)} p' \qquad RT_g(a) = l_{gh}}{p \xrightarrow{(R)\ (\bullet;W;l_{gh};\overline{h}ar\,|\,\Theta)} p'}$$

Now, consider the network depicted in Fig.4. We have three ASs: an Italian one, a German one and an English one, whose gateways are respectively the sites it, de and en; and we have two processes willing to communicate from site a in AS_{it} to site b in AS_{en}. Suppose there is a path from a to it in AS_{it}, the routing tables are such that $RT_{it}(b) = l_{it\,de}$ and $RT_{de}(b) = l'_{de\,en}$, and that there is a path in AS_{en} from en to b. Let G be the process that models forwarding from gateways, namely $L(l_{it\,de}) \mid L(l''_{it\,en}) \mid L(l'_{de\,en})$. Then we can infer

$$AS_{it} \mid AS_{en} \mid AS_{de} \mid G \xrightarrow{\bullet;l_{it\,de};l'_{de\,en};\bullet} AS'_{it} \mid AS'_{en} \mid AS_{de} \mid G \ .$$

Notice that only the part of the path between the gateways is observable.

6 Conclusions

In this paper we presented NCPi, an extension of π-calculus with an explicit notion of network. To achieve this the syntax is enriched with named connectors. From a semantic point of view, an observation is a snapshot of the traffic on the network, represented as the paths concurrently covered by the data. The semantics' concurrent nature is the key feature that allows bisimilarity to be a congruence.

[4] Roles played by sites, such as "gateway", are stated informally here, but they could be formalized through a type system.

6.1 Related Work

The works most closely related to ours are [12] and [9] where network-aware extensions of $D\pi$ [15] and KLAIM [7] are presented, called respectively $D\pi_F$ and TKLAIM. KLAIM is quite far from the synchronous π-calculus, because it models a distributed tuple-space modifiable through asynchronous primitives, but an encoding to the asynchronous π-calculus exists [8]. Both $D\pi_F$ and TKLAIM are *located* process calculi, which means that processes are deployed in *locations*, modeling physical network nodes. In NCPi, instead, processes access the network through sites, possibly more than one for each process, rather than being inside the network. However, locations can be easily introduced in NCPi by a typing mechanism which limits the number of subject names in processes. The network representations are quite different: in $D\pi_F$ locations are explicitly associated with their connectivity via a type system, TKLAIM has a special process to represent connections, while in our calculus connections are just names, so the available network nodes and connections correspond to the standard notion of free names. This brings simpler primitives, but also a higher level of dynamism: connections can be created and passed among processes, as shown in §2; this example, in our opinion, is not easily implementable in TKLAIM and $D\pi_F$. Moreover, our calculus is more programmable: processes explicitly activate transportation services over connections via the link prefix, while in the cited calculi the network is always available.

We can also cite [13,14,10] as examples of calculi where resources carry some extra information: they explicitly associate costs with π-calculus channels through a type system. In our case, links could also be typed in order to model services with different features, e.g. performance, costs and access rights.

6.2 Research Directions

Our calculus only captures point-to-point communication, but a network could be used for more complex forms of interaction, e.g. multicast. One possible development direction might be allowing different mechanisms of message exchanging. Moreover, one can think of complex QoS conditions on resources, e.g. restrictions on bandwidth or costs. There is also some room for asynchronous variations, for instance each hop of a routing path could be performed in different transitions. This would capture the step-by-step behavior of SDNs.

Network-awareness is only one form of resource-awareness, which is essential to adequately model new computational paradigms such as cloud computing. Future work includes also the development of an algebraic/coalgebraic categorical model of resource-aware nominal calculi. In particular, the approach based on presheaf models has been successfuly applied to the π-calculus [11], the fusion calculus [19] and the explicit fusion calculus [2]. This approach is especially effective for nominal calculi, because it allows to model resources as a separate category, so to decouple the structure of resources from the syntax and semantics of processes using them. This permits to capture many alternatives with minimal changes. Moreover, coalgebras over a broad class of presheaves can be implemented as HD-automata [5,4], more concrete operational models that allow for name deallocation and hence are suitable for verification purposes. In our case, the resources of a process are its free sites and links, which can be represented as a finite graph. Functors on the category of resources could allow to create new sites and new links, and to increase their capabilities, similarly to what happens with functor δ in the presheaf semantics of the π-calculus.

References

[1] *Openflow foundation website*, http://www.openflow.org/.

[2] Bonchi, F., M. G. Buscemi, V. Ciancia and F. Gadducci, *A category of explicit fusions*, in: *Concurrency, Graphs and Models*, 2008, pp. 544–562.

[3] Campbell, A. T. and I. Katzela, *Open signaling for atm, internet and mobile networks (opensig'98)* (1999).

[4] Ciancia, V., A. Kurz and U. Montanari, *Families of symmetries as efficient models of resource binding*, Electr. Notes Theor. Comput. Sci. **264** (2010), pp. 63–81.

[5] Ciancia, V. and U. Montanari, *Symmetries, local names and dynamic (de)-allocation of names*, Information and Computation **208** (2010), pp. 1349 – 1367.

[6] Corradini, A., U. Montanari and F. Rossi, *An abstract machine for concurrent modular systems: Charm*, Theor. Comput. Sci. **122** (1994), pp. 165–200.

[7] De Nicola, R., G. Ferrari and R. Pugliese, *Klaim: A kernel language for agents interaction and mobility*, IEEE Trans. Software Eng. **24** (1998), pp. 315–330.

[8] De Nicola, R., D. Gorla and R. Pugliese, *On the expressive power of klaim-based calculi*, Theor. Comput. Sci. **356** (2006), pp. 387–421.

[9] De Nicola, R., D. Gorla and R. Pugliese, *Basic observables for a calculus for global computing*, Information and Computation **205** (2007), pp. 1491 – 1525.

[10] De Vries, E., A. Francalanza and M. Hennessy, *Reasoning about explicit resource management (extended abstract)*, in: *PLACES: Programming Language Approaches to Concurrency and Communication-cEntric Software*, ETAPS, 2011, pp. 15–21, http://places11.di.fc.ul.pt/.

[11] Fiore, M. P. and D. Turi, *Semantics of name and value passing*, in: *LICS*, 2001, pp. 93–104.

[12] Francalanza, A. and M. Hennessy, *A theory of system behaviour in the presence of node and link failure*, Information and Computation **206** (2008), pp. 711 – 759.

[13] Hennessy, M., *A calculus for costed computations*, Logical Methods in Computer Science **7** (2011).

[14] Hennessy, M. and M. Gaur, *Counting the cost in the picalculus (extended abstract)*, Electr. Notes Theor. Comput. Sci. **229** (2009), pp. 117–129.

[15] Hennessy, M. and J. Riely, *Resource access control in systems of mobile agents*, Inf. Comput. **173** (2002), pp. 82–120.

[16] Lanese, I., "Synchronization strategies for global computing models," Ph.D. thesis, Computer Science Department, University of Pisa, Pisa, Italy (2006).

[17] Lanese, I., *Concurrent and located synchronizations in i-calculus*, in: J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack and F. Plasil, editors, *SOFSEM (1)*, Lecture Notes in Computer Science **4362** (2007), pp. 388–399.

[18] McKeown, N., T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker and J. S. Turner, *Openflow: enabling innovation in campus networks*, Computer Communication Review **38** (2008), pp. 69–74.

[19] Miculan, M., *A categorical model of the fusion calculus*, Electr. Notes Theor. Comput. Sci. **218** (2008), pp. 275–293.

[20] Milner, R., “A Calculus of Communicating Systems,” Lecture Notes in Computer Science **92**, Springer, 1980.

[21] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, i*, Inf. Comput. **100** (1992), pp. 1–40.

[22] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, ii*, Inf. Comput. **100** (1992), pp. 41–77.

[23] Montanari, U. and M. Sammartino, *Network conscious pi-calculus*, Technical Report TR-12-01, Computer Science Department, University of Pisa (2012).

[24] Tennenhouse, D. L. and D. J. Wetherall, *Towards an active network architecture*, Computer Communication Review **26** (1996), pp. 5–18.

[25] Y.Rekhter, *A border gateway protocol 4 (bgp-4)*, http://www.ietf.org/rfc/rfc1771.txt (1995).