Spatial-Behavioral Types for Concurrency and Resource Control in Distributed Systems

Luís Caires

CITI / Departamento de Informática, Universidade Nova de Lisboa, Portugal

Abstract

We develop a notion of spatial-behavioral typing suitable to discipline concurrent interactions and resource usage in distributed object systems. Our type structure reflects a resource sensitive model, where a parallel composition type operator expresses resource independence, a sequential composition type operator expresses resource synchronization, and a type modality expresses resource ownership. We model the intended computational systems using a concurrent object calculus. Soundness of our type system is established using a logical relations technique, building on a interpretation of types as properties expressible in a spatial logic.

1 Introduction

The aim of this work is to study typing disciplines for distributed service-based systems, with a particular concern on the key aspects of concurrency, resource control, and compositionality. For our current purposes, we consider service-based systems to be certain kinds of distributed object systems, but where binding between parties is dynamic rather than static, and the fundamental abstraction mechanism is task composition, rather than just remote method invocation. In this paper, we approach the issue of compositional analysis of distributed services and resources using a new notion of typing inspired by spatial logics. Technically, we proceed by introducing a process calculus for distributed services, where clients and servers are represented by concurrent "objects" (aggregates of operations and state). Services are called by reference, and references (names) to services may be passed around, as in π-calculi. New services may also be dynamically instantiated. We then develop and study a fairly expressive type system aimed at disciplining interactions and resource usage in these kind of systems.

Our type structure is motivated by fundamental properties of the intended models. We conceive a service-based system as a layered concurrent and distributed system, where service provider objects execute tasks on behalf of client objects, in a coordinated way. Even if the same object may act as client and server, we do not expect intrinsic cyclic dependencies to occur in such a system. The main coordination abstractions for assembling tasks into services are probably parallel (independent) and sequential composition. Tasks are independent when they never get to compete for resources; independent tasks appear to run simultaneously, this is the default behavior of the "global computer". On the other hand, causality, data flow, and resource competition introduce constraints in the control flow of computations. We will thus consider tasks and resources as the basic building blocks of service based systems.

Models of concurrent programming usually introduce two kinds of entities in their conceptual universe: processes (active) and resources (passive). While processes are a main subject of analysis, resources are considered primitive, further unspecified entities. An essential characteristic of resources is that may not be shared by different processes, by definition (objects such as as files, memory cells, are typical examples). We adopt a different view, according to which resources and objects are not a priori modeled by different sorts of entities (*everything is an object*). Our distinction criteria is observational: what distinguishes a resource among other objects is that resources must be used "with care" so to avoid meaningless or disrupted computations. For example, a massively replicated service (*e.g.*, Google) behaves pretty much as if every client owned its own private copy of it. On the other hand, an object managing a web-service session with a user, is certainly not supposed to be shared: if other user gets in the middle and interferes with the session things may go wrong! We thus consider the latter more "resource-like" than the former.

Instead of conceiving resources as entities external to the model, for which certain usage policies are postulated (as in, *e.g.* [21]), we think of a resource as any object that *must* be used according to a strict discipline to avoid getting into illegal states. Our semantics realizes such illegal states concretely, as "message not understood" errors, rather than as violations of extraneously imposed policies, as *e.g.*, in [17, 21, 2]. Adopting a deep model of resources as fragile objects allows us to consider sharing constraints much more general than the special cases (all or nothing) usually considered: *e.g.*, at certain stage of an usage protocol a resource might be sharable, while at other stage it may be not. Our uniform approach also naturally supports a computational model where resources may be passed around in transactions, buffered in pools, while ensuring that their capabilities are used consistently, by means of typing.

Our type system, we believe, captures fundamental constraints on resource access arising in general concurrent systems. We introduce the basic operators

$$U, V ::= \mathbf{0} \mid \mathtt{l}(U)V \mid U|V \mid U \wedge V \mid U;V \mid U^{\circ} \mid U \rhd V$$

to which we add a recursion operator (and type variables).

Superficially, the underlying structure may seem close to a behavioral algebra, with parallel and sequential composition, primitive action (method call $\mathtt{l}(U)V$), and a few extra operators. However, our aim now is not just to talk about the behavior of systems, but also about resource distribution, and ownership transfer. For example, the spatial composition type $U \mid V$ states that a service may be used accordingly to U and V by independent clients, one using it as specified by U, the other as specified by V. In particular, the tasks U and V may be activated concurrently. For instance, an object typed by *Travel*

$$\mathit{Travel} \triangleq (\mathtt{flight} \mid \mathtt{hotel}); \mathtt{order}$$

will be able to service the `flight` (we abbreviate $\mathtt{l}(\mathbf{0})\mathbf{0}$ by `l`, and so on) and `hotel` tasks simultaneously and after that (and only after that), the `order` task. The spatial interpretation of $U \mid V$, derived from the process decomposition operations of spatial logics for concurrency [7, 11], implies further consequences, namely that the (distributed) resources used by U and V do not interfere [25] ; this property is important to ensure closure under composition of safety properties of typed systems.

Owned types, written U°, in addition to asserting that a service is usable as specified by U, also require such usage to be completely owned. What does this mean? Owned types discipline the dynamic delegation (or transfer) of resources or service references between interacting partners. For example, an operation typed as $\mathtt{use}(V^\circ)U$, requests ownership of its argument of type V. This means that a client calling $\mathtt{use}(v)$ must have, and will loose at the call, ownership of v (or at least of some V°-typed, $\mid$-separated view, of v). So, any object in possession of a reference of owned type may, for example, cache it in memory, for later use, or even dispose of it. On the other hand, if V is not an owned type, calling an operation typed as *e.g.*, $\mathtt{rent}(V)U$, will ensure that an usage as specified by type V will be performed during the call, and that ownership of v, after the exercise of the usage V, will be retained by the caller (according to some continuation type) instead of being transfered to the callee. A return type U is always implicitely considered to be an owned type, because the callee always looses ownership of (some view of) the returned value. Thus owner types control delegation of resources in a much finer way than the more strict (all or nothing) disciplines provided by usual linear types.

Familiar behavioral types may also be easily expressed. For example, the usage protocol of a file of objects of type V may be specified

$$\mathit{File}(V) \triangleq (\mathtt{open}; (\mathtt{read}()V \wedge \mathtt{write}(V^\circ))^\star; \mathtt{close})^\star$$

where $U^\star \triangleq \mathtt{rec}\ \alpha.(\mathbf{0} \wedge (U; \alpha))$ expresses iteration. By combining recursion with spatial types, we can then introduce shared types. A shared type $!U$ states of an object that it may be used according to an unbounded number of independent sessions, each one conforming to the type U. By combining our type operators, we may specify fine grained shared access protocols, such as the typical "multiple readers/unique writer" access pattern for memory cells:

$$RW(V) \triangleq (!\texttt{read}()V; \texttt{write}(V^\circ))^\star$$

Moreover, our types are related by a flexible subtyping relation. Finally, and crucially, guarantee types $U \triangleright V$ allow us to compose subsystems into larger systems, while preserving the properties ensured by their typings.

The paper is structured as follows. In Section 2, we present our core language and its operational semantics, and some examples. In Section 3 we introduce our basic type system, and prove its soundness. Our proof combines syntactical and semantical reasoning, in the spirit of the logical relations technique, where types are interpreted as properties expressed in a spatial logic. In Section 4, we discuss how to extend our basic system to cover more general forms of sharing. Finally, Section 5 overviews related work and draws some conclusions.

2 A Distributed Object Calculus

In this section, we present the syntax and semantics of our distributed object calculus. We first illustrate its main ingredients with a small example. Define

$$\mathit{Counter} \triangleq n[\texttt{inc}() = \texttt{let}\ x = s?\ \texttt{in}\ s!(x+1) \parallel s\,\langle 0\rangle \parallel\ ]$$

The object *Counter* has the structure $n[M \parallel s \parallel t]$ where M are the object methods, s are the object state elements, and t are the object active threads. In the example the object has a single method, a single state element ($s\,\langle 0\rangle$), and no running threads. Invocation of a method causes a new thread to be spawned. For example, invocation of the `inc` method causes the transition

$$\mathit{Counter} \xrightarrow{n.\texttt{inc}_c()} n[\texttt{inc}() = \cdots \parallel s\,\langle 0\rangle \parallel c\,\langle \texttt{let}\ x = s?\ \texttt{in}\ (s!(x+1); x)\rangle] = C_1$$

where c is a freshly created thread identifier. After being created, a thread starts running autonomously, so we have the reduction sequence

$$\begin{aligned} C_1 &\longrightarrow n[\texttt{inc}() = \cdots \parallel s\,\langle 0\rangle \parallel c\,\langle \texttt{let}\ x = 0\ \texttt{in}\ (s!(x+1); x)\rangle] \\ &\longrightarrow n[\texttt{inc}() = \cdots \parallel s\,\langle 0\rangle \parallel c\,\langle s!(1); 0\rangle] \\ &\longrightarrow n[\texttt{inc}() = \cdots \parallel s\,\langle 1\rangle \parallel c\,\langle 0\rangle] = C_2 \end{aligned}$$

At this point, the expression in the thread c has reduced to a value: the thread may now terminate after returning the result back to the caller.

$$C_2 \xrightarrow{n.c(0)} n[\texttt{inc}() = \cdots \parallel s\,\langle 1\rangle \parallel\ ]$$

Our calculus is distributed in the sense that a method call originating at an object site spawns a new thread in the callee site, in general a different site from the caller site. This (quite realistic) behavior is modeled directly in our calculus, allowing us to maintain a substantial degree of locality, which seems necessary for the interpretation of types. In particular, any code acting on the local state of an object is to be found "near" the object state representation in the spatial structure of a configuration, even if it belongs to a system wide transaction, and any method call is processed by means of a call-reply two-message protocol. We may also easily model in the calculus manipulation of remote object references, concurrent method invocation, and state and history dependent computations, as will be demonstrated in forthcoming examples.

Assume given an infinite set $\mathcal{N}$ of *names*, used to identify objects (n, m, p), threads (b, c, d), and state elements in objects (a). We also assume given an infinite set $\mathcal{X}$ of *variables* (x, y, z), and an infinite set $\mathcal{L}$ of *method labels* $(\mathtt{j}, \mathtt{k}, \mathtt{l})$. We note $\mathcal{X} = \mathcal{N} \cup \mathcal{V}$, and let η range over $\mathcal{X}$ (variables and names).

Definition 2.1 (Values, Expressions, Methods) *The sets $\mathcal{V}$ of values, $\mathcal{E}$ of expressions, and $\mathcal{M}$ of methods are defined in Fig. 1 (top).*

Notice that expressions may only syntactically occur either in the body of a method definition, or in a thread. We use the notation $\overline{\varsigma}$ to denote a sequence of syntactical elements of class ς. The value `nil` stands for the null object reference. The method *call* expression $n.\mathtt{l}(v)$ denotes the invocation of the method `l` of object n, where the value v is passed as argument. The *wait* expression $n.c()$ denotes waiting for a reply to a previously issued method invocation of the form $n.\mathtt{l}(v)$, where c is the identifier of the thread which is serving the request remotely. The *wait* construct plays a key technical role in our formulation of the dynamic and static semantics of our language, even if it is not expected to appear in source programs. The *composition* construct $\mathtt{let}\ \overline{x = e}\ \mathtt{in}\ f$ denotes the parallel evaluation of the expressions e_i, followed by the evaluation of the body f, where the result of evaluating each e_i is bound to the corresponding x_i. The x_i are distinct bound variables, with scope the body f. The `let` construct allows us to express arbitrary parallel / sequential control flow graphs, in which values may be propagated between parallel and sequential subcomputations. We then use the following abbreviations (where x_1 and x_2 do not occur in e_1 and e_2):

$$(e_1 \mid e_2) \triangleq \mathtt{let}\ x_1 = e_1\ ,\ x_2 = e_2\ \mathtt{in\ nil} \qquad (e_1; e_2) \triangleq \mathtt{let}\ x_1 = e_1\ \mathtt{in}\ e_2$$

The $a?$ and $a!(v)$ constructs allow objects to manipulate their local store. The *read* expression $a?$ returns a value stored under tag a, while the *write* expression $a!(v)$ stores value v in the store under tag a. The store conforms to a resource space, where reading consumes data, and writing replaces existing with new data elements. Evaluation of $\mathtt{new}[\overline{a}; M]$ results in the allocation of

$$
\begin{array}{rlll}
e ::= & f,g,h \in \mathcal{E} & & \text{(Expressions)} \\
 & v & & \text{(Value)} \\
\mid & v.\mathtt{l}(v) & & \text{(Call)} \\
\mid & n.c() & & \text{(Wait)} \\
\mid & a? & & \text{(Read)} \\
\mid & a!(v) & & \text{(Write)} \\
\mid & \mathtt{new}\ [\overline{a}; M] & & \text{(Instantiation)} \\
\mid & \mathtt{let}\ \overline{x = e}\ \mathtt{in}\ e & & \text{(Composition)} \\
\mid & \mathtt{rec}\ x.e & & \text{(Recursion)}
\end{array}
\qquad
\begin{array}{rlll}
v ::= & r \in \mathcal{V} & & \text{(Values)} \\
 & n & & \text{(Name)} \\
\mid & x & & \text{(Identifier)} \\
\mid & \mathtt{nil} & & \text{(Termination)} \\
 & & & \\
M ::= & \in \mathcal{M} & & \text{(Methods)} \\
 & \mathbf{0} & & \text{(Empty)} \\
\mid & \mathtt{l}(x) = e & & \text{(Method)} \\
\mid & M \mid M & & \text{(Methods)}
\end{array}
$$

$$
\begin{array}{rll}
s ::= & \in \mathcal{S} & \text{(Stores)} \\
 & \mathbf{0} & \\
\mid & a\langle v\rangle & \\
\mid & s \mid s &
\end{array}
\quad
\begin{array}{rll}
t ::= & \in \mathcal{T} & \text{(Threads)} \\
 & \mathbf{0} & \\
\mid & t \mid t & \\
\mid & c\langle e\rangle &
\end{array}
\quad
\begin{array}{rll}
P ::= & Q,R \in \mathcal{P} & \text{(Networks)} \\
 & \mathbf{0} & \\
\mid & (\boldsymbol{\nu} n)P & \\
\mid & P \mid Q & \\
\mid & n[M \parallel s \parallel t] & \text{(Object)}
\end{array}
$$

Fig. 1. Values, Expressions, Methods, Stores, Threads, Networks.

$$
n.\mathtt{l}(v) \xrightarrow{\overline{n.\mathtt{l}_c(v)}} n.c()
\qquad
\mathtt{new}\,[\overline{a}; M] \xrightarrow{n[\overline{a};M]} n
$$

$$
n.c() \xrightarrow{n.c(v)} v
\qquad
\frac{e\{{}^{x}/_{\mathtt{rec}\ x.e}\} \xrightarrow{\alpha} e'}{\mathtt{rec}\ x.e \xrightarrow{\alpha} e'}
$$

$$
a? \xrightarrow{a?(v)} v
\qquad
\frac{e \xrightarrow{\alpha} e'}{\mathtt{let}\ \cdots, x = e, \cdots\ \mathtt{in}\ f \xrightarrow{\alpha} \mathtt{let}\ \cdots, x = e', \cdots\ \mathtt{in}\ f}
$$

$$
a!(v) \xrightarrow{a!(v)} \mathtt{nil}
\qquad
\mathtt{let}\ \overline{x = v}\ \mathtt{in}\ e \xrightarrow{\tau} e\overline{\{{}^{x}/_{v}\}}
$$

Fig. 2. Evaluation (Expressions).

a new object, with set of methods M, and whose identity (a fresh name) is returned; $\overline{a}$ declares the object local state. In the *method* $\mathtt{l}(x) = e$ the parameter x is bound in the scope of the method body e (for the sake of simplicity, we just consider a single parameter). Finally, the `rec` construct introduces recursion. To keep our language "small", we refrain from introducing other useful ingredients, such as basic data types and related operators, for instance booleans and conditionals. It should be easy to formally extend the language with such constructs, so we will sometimes use them (*e.g.*, in examples).

Example 2.2 *We sketch a toy scenario of service composition, where several sites cooperate to provide a travel booking service. First, there is an object*

F implementing a service for finding and booking flights. It provides three methods: `flight` *to look for and reserve a flight,* `book` *to commit the booking, and* `free` *to release a reservation. A similar service is provided by object H, used for booking hotel rooms.*

$$
\begin{aligned}
F &\triangleq f[\,\texttt{flight}() = \cdots \mid \texttt{book}() = \cdots \mid \texttt{free}() = \cdots \parallel \ \parallel]\\
H &\triangleq h[\quad \texttt{hotel}() = \cdots \mid \texttt{book}() = \cdots \mid \texttt{free}() = \cdots \parallel \ \parallel]\\
G &\triangleq gw[\,\texttt{pay}(s) = \texttt{if}\ bk.\texttt{debit}()\ \texttt{then}\ s.\texttt{book}()\ \texttt{else}\ s.\texttt{free}() \parallel \ \parallel]\\
B &\triangleq br[\,\texttt{flight}() = f.\texttt{flight}() \mid \texttt{hotel}() = h.\texttt{hotel}() \mid\\
&\qquad\quad \texttt{order}() = (gw.\texttt{pay}(f); gw.\texttt{pay}(h)) \parallel \ \parallel]
\end{aligned}
$$

We elide method implementations in F and G, but assume that the operations must be called in good order to avoid disruption, namely that after calling `flight`*, a client is supposed to call either* `book` *or* `free`*. The broker B, that implements the front-end of the whole system, is client of F and H, and also of a payment gateway G. The gateway books items if succeeds in processing their payment through a remote bank service named bk. Our travel booking service, available at br, is used by first invoking the* `flight` *and* `hotel` *operations in any order. In fact, these operations may be called concurrently, since they trigger separate computations. Afterwards, the* `order` *operation may be invoked to book and pay for both items, delegating access to f and h to the gateway. The session will then terminate, and the broker becomes ready for another round. We will see in this paper how usage patterns such as these may be specified by typing, and how the type of a whole system may be compositionally defined from the types of its components.*

Definition 2.3 (Stores, Threads, Networks) *The sets $\mathcal{S}$ of stores, $\mathcal{T}$ of threads, and $\mathcal{P}$ of networks are given in Fig. 1 (bottom).*

A network is a (possibly empty) composition of objects, where composition $P \mid Q$ and restriction $(\boldsymbol{\nu} n)P$ are introduced with their usual meaning (cf., the π-calculus). An object $n[M \parallel s \parallel t]$ encapsulates, under the object name n, some methods M (passive code), a store s (that holds the object local state), and some threads t (active, running code). A store s is a bag of pairs tag - value. Each value is recorded in a store under an access tag (a name), represented by $a\langle v\rangle$, where a is the tag and v is the value. On the other hand, a thread $c\langle e\rangle$ is uniquely identified by its name c and holds an active code fragment, namely the expression e. As already mentioned, threads are spawned when methods are called, and may run concurrently with other independent threads in the same object or network.

For any sets of names S, S', we write $n\#S$ (resp. $S\#S'$) to denote that $n \notin S$ (resp. that S and S' are disjoint). We use A, B, C to range over $\mathcal{M} \cup \mathcal{S} \cup \mathcal{T} \cup \mathcal{P}$ (that is, over all entities which are "composable" under $\mid$).

By *fn*(P) (resp. *fn*(t), *fn*(s), etc.) we denote the set of free names in process P (resp. thread t, store s, etc.), defined as expected. We also define by *ft*(P) the set of free thread names in P, by *lb*(P, n) the set of method labels of object n in P, and by *st*(P, n) the set of store tags of object n in P:

$$\begin{array}{ll} c \in \mathit{ft}(P) & \textit{iff } P \equiv (\boldsymbol{\nu}\overline{m})(n[\ \|\ \ \|\ c\langle e\rangle] \mid Q) \textit{ and } c\#\overline{m} \\ \mathtt{l} \in \mathit{lb}(P,n) & \textit{iff } P \equiv (\boldsymbol{\nu}\overline{m})(n[\mathtt{l}(x) = e \ \|\ \ \|\ ] \mid Q) \textit{ and } n\#\overline{m} \\ a \in \mathit{st}(P,n) & \textit{iff } P \equiv (\boldsymbol{\nu}\overline{m})(n[\ \|\ a\,\langle v\rangle\ \|\ ] \mid Q) \textit{ and } n\#\overline{m} \end{array}$$

Any object in a system is given a unique name, so that, for instance, the network term $n[M \parallel s \parallel t] \mid n[N \parallel r \parallel u]$ denotes the same network as the term $n[M \mid N \parallel s \mid r \parallel t \mid u]$. Spatial identities such as this one (the **Split** law) are formally captured by structural congruence, defined below. All axioms are familiar from π-calculi, except the split law [17, 24], that allows individual objects to be split up to the parallel composition operator $\mid$.

Definition 2.4 (Structural Congruence) *Structural congruence, noted* $\equiv$, *is the least congruence relation on networks, methods, and threads, such that*

$$\begin{array}{rr} A \equiv A \mid \mathbf{0} & (\boldsymbol{\nu}n)(P \mid Q) \equiv P \mid (\boldsymbol{\nu}n)Q \ \ \textit{if } n\#\mathit{fn}(P) \\ B \mid A \equiv A \mid B & A \mid (B \mid C) \equiv (A \mid B) \mid C \\ (\boldsymbol{\nu}n)\mathbf{0} \equiv \mathbf{0} & (\boldsymbol{\nu}m)(\boldsymbol{\nu}n)P \equiv (\boldsymbol{\nu}n)(\boldsymbol{\nu}m)P \\ & n[M \parallel s \parallel t] \mid n[N \parallel r \parallel u] \equiv n[M \mid N \parallel s \mid r \parallel t \mid u] \end{array}$$

We also define the partial order $\leqq$ *by* $P \leqq Q \triangleq$ *exists* R . $Q \equiv R \mid P$.

To lighten our notation, we avoid writing $\mathbf{0}$ in object slots, leaving blank the corresponding place . *E.g.*, $n[M \parallel \mathbf{0} \parallel \mathbf{0}]$ will be written simply as $n[M \parallel\ \ \parallel\ ]$.

The operational semantics of networks is defined by suitable transition relations: we define a transition system specifying the evaluation of expressions (in Fig. 2), and another transition system to specify network reduction (in Fig. 3). Transition system are labeled: the various labels in expression transitions express the various kinds of actions a running thread may perform.

Definition 2.5 *Labels* $\mathcal{L}$ *are given by:*

$$\alpha ::= \boldsymbol{\tau} \mid \overline{n.\mathtt{l}_c(v)} \mid n.\mathtt{l}_c(v) \mid n.c(v) \mid \overline{n.c(v)} \mid a?(v) \mid a!(v) \mid n[\overline{a}; M]$$

We have internal computation ($\boldsymbol{\tau}$), method call ($\overline{n.\mathtt{l}_c(v)}$), wait for method reply ($n.c(v)$), reading to the state ($a?(v)$), writing to the state ($a!(v)$). A $n[\overline{a}; M]$ labeled transition, caused by the evaluation of a `new` $[\overline{a}; M]$ expression, signals the creation of a new object.

$$\frac{M \equiv \mathtt{l}(x) = h \mid N \qquad e \xrightarrow{\overline{n.\mathtt{l}_c(v)}} e' \qquad [c\ \mathit{fresh}]}{n[M \parallel\ \parallel\ ] \mid m[\ \parallel\ \parallel b\langle e\rangle] \to (\boldsymbol{\nu} c)(n[M \parallel\ \parallel c\langle h\{^x/_v\}\rangle] \mid m[\ \parallel\ \parallel b\langle e'\rangle])}$$

$$\frac{e \xrightarrow{n.c(r)} e'}{n[\ \parallel\ \parallel c\langle r\rangle] \mid m[\ \parallel\ \parallel b\langle e\rangle] \to m[\ \parallel\ \parallel b\langle e'\rangle]} \qquad \frac{e \xrightarrow{\boldsymbol{\tau}} e'}{n[\ \parallel\ \parallel c\langle e\rangle] \to n[\ \parallel\ \parallel c\langle e'\rangle]}$$

$$\frac{e \xrightarrow{a?(v)} e'}{n[\ \parallel a\,\langle v\rangle\ \parallel c\langle e\rangle] \to n[\ \parallel a\langle \mathtt{nil}\rangle\ \parallel c\langle e'\rangle]} \qquad \frac{e \xrightarrow{a!(v)} e'}{n[\ \parallel a\,\langle u\rangle\ \parallel c\langle e\rangle] \to n[\ \parallel a\langle v\rangle\ \parallel c\langle e'\rangle]}$$

$$\frac{e \xrightarrow{m[\overline{a};M]} e' \qquad [m\ \mathit{fresh}]}{n[\ \parallel\ \parallel c\langle e\rangle] \to (\boldsymbol{\nu} m)(m[M \parallel \overline{a\langle \mathtt{nil}\rangle} \parallel\ ] \mid n[\ \parallel\ \parallel c\,\langle e'\rangle])}$$

$$\frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q} \qquad \frac{P \to Q}{(\boldsymbol{\nu} n)P \to (\boldsymbol{\nu} n)Q} \qquad \frac{P \to Q}{P \mid R \to Q \mid R}$$

Fig. 3. Reduction (Networks).

We comment on the key rules in Fig. 2. A remote method call reduces to a wait expression on a fresh thread name c. Such wait expression will reduce to the returned value upon thread termination. The `let`-introduced sub-expressions are evaluated concurrently, until each one reduces to a value; only after that the `let` body is activated. The transition system in Fig. 3 specifies a reduction relation on networks, modeling our intended remote method call protocol. Servicing a method call causes a new thread to be spawned at the callee object's location, to execute the method's body. At that point, the thread that originated the call suspends, waiting for a reply. Such a reply will be sent back to the caller, after the servicing thread terminates.

Definition 2.6 (Evaluation and Reduction) Evaluation*, noted* $e \xrightarrow{\alpha} e'$*, is the relation defined on expressions by the labeled transition system in Fig. 2.* Reduction*, noted* $P \to Q$*, is the relation defined on networks by the transition system in Fig. 3.*

We write $\Rightarrow$ for the the reflexive transitive closure of $\to$. Notice the role of structural congruence in reduction, in particular the **Split** law, so that each rule may mention just the relevant parts of objects for each interaction case. Besides reduction, it is useful to introduce a labeled transition system for networks, extending the reduction semantics with labels in order to capture incoming method calls from the environment, and replies to them.

Definition 2.7 (Labeled Transition System) *The labeled transition relation on networks is the least relation defined by the rules:*

$$\frac{P \to Q}{P \xrightarrow{\tau} Q} \qquad \frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q}$$

$$\frac{[c \textit{ fresh and } n, u \# \overline{m}]}{(\boldsymbol{\nu}\overline{m})(n[\mathtt{l}(x) = e \parallel \ \parallel\ ] \mid R) \xrightarrow{n.\mathtt{l}_c(u)} (\boldsymbol{\nu}\overline{m})(n[\mathtt{l}(x) = e \parallel \ \parallel c\langle e\{^x/_u\}\rangle] \mid R)}$$

$$\frac{[c, n \# \overline{m}]}{(\boldsymbol{\nu}\overline{m})(n[\ \parallel \ \parallel c\langle r\rangle] \mid R) \xrightarrow{\overline{n.c(r)}} (\boldsymbol{\nu}\overline{m} \setminus r)(n[\ \parallel \ \parallel\ ] \mid R)}$$

The first two rules incorporate reduction as silent transition, as usual. The third rule captures an incoming method call from the environment. The last rule captures the reply to the environment of a pending method call, upon thread termination. Notice that the labeled transition system in Definition 2.7 formalizes the transitions presented in the counter example at the very beginning of this section. We thus conclude the technical presentation of our distributed object calculus.

Before closing the section, we discuss some details of our model, and introduce several useful auxiliary concepts. The operational semantics we have defined assumes and preserves certain regularity conditions on object networks. In general, a network P is said to be *well-formed* if all threads occurring in P have distinct names, all methods in the same object have distinct labels, all state elements in the same object have distinct tags, and for any thread name c there is at most one occurrence in P of a wait expression $n.c()$. We have

Lemma 2.8 (Preservation of well-formedness) *If P is a well-formed network and $P \xrightarrow{\alpha} Q$ then Q is also a well-formed network.*

An object may only become active as effect of an incoming method call issued by a running thread. An object $n[M \parallel s \parallel t]$ such that $t \equiv \mathbf{0}$, is said to be *idle*, since it contains no running threads. Likewise, a network is idle if all of its objects are idle. We define:

Definition 2.9 $\text{idle}(P) \triangleq$ For all Q.*if* $P \equiv (\boldsymbol{\nu}\overline{m})(n[;;t] \mid Q)$ *then* $t \equiv \mathbf{0}$.

Even well-formed networks may get *stuck* if a method call is outstanding, but the called object does not offer the requested method. An attempt to read from or write to an undefined store element can also cause an object to get stuck. We define:

Definition 2.10 $\text{stuck}(P) \triangleq$ exists $\overline{m}, Q, e, e'$. $P \equiv (\boldsymbol{\nu}\overline{m})(p[\ \parallel \ \parallel c\,\langle e\rangle] \mid Q)$ *and either* $e \xrightarrow{\overline{n.\mathtt{l}_c(v)}} e'$ *and* $\mathtt{l} \notin lb(Q, n)$*, or* $e \xrightarrow{a-(v)} e'$ *and* $a \notin st(Q, p)$*.*

Networks, as modeled in our calculus, may easily get stuck, if not carefully designed. As in more familiar untyped object oriented programming languages, *message-not-understood* errors may arise whenever an object does not implement the invoked method. However, in our present context stuck states may also arise if method calls are not coordinated and timing errors occur. For example, such situations may occur when clients do not respect protocols, or when race conditions arise, *e.g.*, during method calls to the same non-shareable method. Moreover, the presence of state in objects creates history dependencies on resource usage, and introduces a grain of resource sensitiveness in our model, as discussed in the Introduction, and illustrated in our next example.

Example 2.11 *Consider the object S defined thus*

$$\begin{aligned} S \triangleq \mathit{server}[\ & \texttt{init}() = s!(\texttt{nil}) \\ & \texttt{open}() = \texttt{let}\ r = \mathit{pool}.\texttt{alloc}()\ \texttt{in}\ s!(r) \\ & \texttt{use}() = \texttt{let}\ r = s?\ \texttt{in}\ (r.\texttt{use}(); s!(r)) \\ & \texttt{close}() = \texttt{let}\ x = s?\ \texttt{in}\ (\mathit{pool}.\texttt{free}(x); s!(\texttt{nil}))\ \|\ \ \|\ ] \end{aligned}$$

The object S is a spooler that offers (some further unspecified) service by relying on a remote resource pool object to fetch appropriate service providers. All service providers (e.g., printers, seen as resources) held in the pool (by reference, of course) are modeled also as objects. Each one of such objects is then assumed to implement the operation of interest `use`*. The server provides the* `use` *service repeatedly to a given client, by forwarding it through a locally cached reference to some allocated service provider (stored in $s\langle - \rangle$). First, the server is initialized: by calling the* `init` *method the local reference is set to* `nil`*. Afterwards, a client must open the service by calling the* `open` *method before using it (so that the server can acquire an available resource), and close it after use by calling the* `close` *method (so that the server may release the resource). The server implements these operations by accessing the pool through its* `alloc` *and* `free` *methods. The internal state of the server, hidden to clients, will always be either of the form $s\langle\texttt{nil}\rangle$, or $s\langle r\rangle$ where r is a reference (a name) of an allocated resource (some object). Notice how the idiom $\texttt{let}\ r = s?\ \texttt{in}\ (\cdots; s!(r))$ is used to express retrieving the value r from the local state, using it (in the $\cdots$ part), and storing it back again.*

The key morale in this example is that the usage protocols described above for the various objects in the scene must be strictly followed to avoid runtime errors, in particular due to resource non-availability. This would occur, e.g., *if the* `use` *operation is invoked right after* `close`*, an attempt to call the* `use` *method on a* `nil` *reference will cause the system to get stuck (possibly causing a crash or a deadlock in a real system).*

A main motivation for our type system, presented in detail in forthcoming sections of this paper, is to prevents erroneous behaviors such as the one illustrated above, by ensuring that all services in a network conform to well-defined concurrency and resource usage protocols.

Fact 1 *If P is well-defined and $P \equiv Q \mid R$ then both Q and R are well-defined.*

However, the converse property does not hold in general, since P and Q might clash in method or thread names. We then define a predicate $(P \| Q)$ to assert that P and Q are composable, in the sense that if P and Q are both well-defined and $P \| Q$ holds, then $P \mid Q$ is also well-defined.

Definition 2.12 (Composable Networks)

$$P \| Q \quad \triangleq \quad ft(Q)\#ft(P) \text{ and forall } n.\ lb(P,n)\#lb(Q,n) \text{ and } st(P,n)\#st(Q,n)$$

Notice that $P \| Q$ *does not* necessarily imply $fn(P)\#fn(Q)$. Henceforth, we assume networks to be always well-defined.

3 Spatial-Behavioral Types

In this section, we present in detail our notion of spatial-behavioral type, and develop a type system to discipline interactions between distributed objects modeled in the calculus presented above. Intuitively, a type T describes a usage pattern for a given object. An assertion of the form $n : T$ states that the object named n may be safely used as specified by the type T. In general, the type of a network P is expressed by a composite assertion $n_1 : T_1 \mid \ldots \mid n_k : T_k$ that specifies the types of various objects named $n_1, \ldots, n_k$ available in P to the external environment. Such a typing environment states that the system provides *independent* services at the names n_i, each one able to be safely used as specified by the type T_i respectively. We first introduce the syntax of types.

Definition 3.1 (Types) *The set $\mathcal{T}$ of types is inductively defined by the following abstract syntax:*

$$\begin{array}{llllll}
T ::= & U, V \in \mathcal{T} & \textit{(Types)} & & & \\
& \mathbf{0} & \textit{(}\mathbf{0}\textit{)} & \mid T \mid U & \textit{(Spatial Composition)} \\
\mid & T \wedge U & \textit{(Conjunction)} & \mid T; U & \textit{(Sequential Composition)} \\
\mid & T^{\circ} & \textit{(Owned)} & \mid \mathtt{l}(U)V & \textit{(Method)} \\
\mid & \alpha & \textit{(Variable)} & \mid \mathtt{rec}\ \alpha.T & \textit{(Recursion)}
\end{array}$$

We first explain the intuitive meaning of the various kinds of types, by interpreting them as properties of objects.

- An object satisfies $n : \mathbf{0}$ if it is idle (Definition 2.9).
- An object satisfies $n : T \mid U$ if it can independently satisfy both $n : T$ and $n : U$. We may also understand such a typing as the specification of two independent views for the object n. More precisely, a $n : T \mid U$ typing says

that the interfaces T and U provided by object n are based in disjoint (in a sense to be made precise below) sets of resources / subsystems, and thus may be safely invoked concurrently.

- An object satisfies $n : T \wedge U$ if it can satisfy both $n : T$ and $n : U$, although not necessarily concurrently. Conservatively, such an object may only be used either as specified by $n : T$ or as specified by $n : U$, being the choice made by the object's client.
- An object satisfies $n : T; U$ if it can satisfy first $n : T$ and afterwards $n : U$, in sequence. In particular, it will only be obliged to satisfy $n : U$ after being used as specified by $n : T$. Implicit in this description is the notion of "usage according to a type", and "termination" of such an usage; we will get back to this point later.
- As explained in the Introduction, the owned type $n : T^\circ$ means that the object may be used as specified by T, but furthermore (and crucially) that this T view must be *exclusively owned*. For example, a reference of type $n : T^\circ$ may be stored in the local state of an object, or returned by a method call, although a reference of type $n : T$ may not, because of possible liveness constraints associated to the type T. This will become clearer in the precise semantic definitions below.
- An object satisfies $n : \mathtt{l}(U)V$ if it offers a method `l` that whenever passed an argument of type U is ensured to return back a result of type V°, and exercise, during the call, an usage of the argument conforming to type U. Thus, a method type specify both a safety and a liveness properties. Notice also that the result is always an owned type: this reflects the fact that an object cannot both retain exclusive ownership of a reference and return it at the end of a method call.
- Recursive types are interpreted as greatest fixed points, we will not detail the developments related to recursion, as they would follow predictable lines.

We now enter the technical description of our type system. A typing environment $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \sigma, \delta)$ is a finite partial mapping from $\mathcal{N} \cup \mathcal{V}$ to $\mathcal{T}$. We write $\mathbf{A} \triangleq \eta_1 : T_1, \ldots, \eta_n : T_n$ for the typing environment $\mathbf{A}$ with domain $\mathfrak{D}(\mathbf{A}) = \{\eta_1, \ldots, \eta_n\}$ such that $\mathbf{A}(\eta_i) = T_i$, for $i = 1, \ldots, n$. We extend type operations $\mathbf{0}$, $(T \mid U)$, $(T \wedge U)$, $(T; U)$ and T° to typing environments pointwise, as follows. $\mathbf{0}$ denotes any typing environment (including the empty one) that assigns $\mathbf{0}$ to all elements in its domain. Given $\mathbf{A}$ and $\mathbf{B}$ such that $\mathfrak{D}(\mathbf{A}) = \mathfrak{D}(\mathbf{B})$, we define type environments $\mathbf{0}$, $(\mathbf{A} \mid \mathbf{B})$, $(\mathbf{A}; \mathbf{B})$, $(\mathbf{A} \wedge \mathbf{B})$, and $\mathbf{A}^\circ$, all with domain $\mathfrak{D}(\mathbf{A})$, such that, for all $\eta \in \mathfrak{D}(\mathbf{A})$, we have

$$\mathbf{0}(\eta) \triangleq \mathbf{0} \qquad (\mathbf{A} \mid \mathbf{B})(\eta) \triangleq \mathbf{A}(\eta) \mid \mathbf{B}(\eta) \qquad (\mathbf{A}; \mathbf{B})(\eta) \triangleq \mathbf{A}(\eta); \mathbf{B}(\eta)$$
$$(\mathbf{A} \wedge \mathbf{B})(\eta) \triangleq \mathbf{A}(\eta) \wedge \mathbf{B}(\eta) \qquad \mathbf{A}^\circ(\eta) \triangleq \mathbf{A}(\eta)^\circ$$

Given a sequence $\overline{T} = T_1, \ldots, T_n$ of types (or typing environments) we denote by $\Pi(\overline{T})$ the type (or typing environment) $(T_1 \mid \cdots \mid T_n)$. When we write $\mathbf{A}, \mathbf{B}$ for a type environment we mean that the domains of $\mathbf{A}$ and $\mathbf{B}$ are disjoint.

$$\overset{(\text{ Refl })}{\mathbf{A} <: \mathbf{A}} \qquad \overset{(\text{ SeqIdL })}{\mathbf{0}; \mathbf{A} <:> \mathbf{A}} \qquad \overset{(\text{ SeqIdR })}{\mathbf{A}; \mathbf{0} <:> \mathbf{A}} \qquad \overset{(\text{ ParId })}{\mathbf{0} \mid \mathbf{A} <:> \mathbf{A}} \qquad \overset{(\text{ ParComm })}{\mathbf{A} \mid \mathbf{B} <: \mathbf{B} \mid \mathbf{A}}$$

$$\overset{(\text{ SeqAssoc })}{\mathbf{A}; (\mathbf{B}; \mathbf{C}) <:> (\mathbf{A}; \mathbf{B}); \mathbf{C}} \qquad \overset{(\text{ ParAssoc })}{(\mathbf{A} \mid \mathbf{B}) \mid \mathbf{C} <:> \mathbf{A} \mid (\mathbf{B} \mid \mathbf{C})}$$

$$\overset{(\text{ ParSeq })}{(\mathbf{A}; \mathbf{B}) \mid (\mathbf{C}; \mathbf{D}) <: (\mathbf{A} \mid \mathbf{C}); (\mathbf{B} \mid \mathbf{D})}$$

$$\overset{(\text{ AndL })}{\mathbf{A} \wedge \mathbf{B} <: \mathbf{A}} \qquad \overset{(\text{ AndR })}{\mathbf{A} \wedge \mathbf{B} <: \mathbf{B}} \qquad \overset{(\text{ And })}{\frac{\mathbf{A} <: \mathbf{B} \quad \mathbf{A} <: \mathbf{C}}{\mathbf{A} <: \mathbf{B} \wedge \mathbf{C}}}$$

$$\overset{(\text{ Trans })}{\frac{\mathbf{A} <: \mathbf{B} \quad \mathbf{B} <: \mathbf{C}}{\mathbf{A} <: \mathbf{C}}} \qquad \overset{(\text{ ParCong })}{\frac{\mathbf{A} <: \mathbf{B}}{\mathbf{A} \mid \mathbf{C} <: \mathbf{B} \mid \mathbf{C}}} \qquad \overset{(\text{ SeqCongL })}{\frac{\mathbf{A} <: \mathbf{B}}{\mathbf{A}; \mathbf{C} <: \mathbf{B}; \mathbf{C}}} \qquad \overset{(\text{ SeqCongR })}{\frac{\mathbf{A} <: \mathbf{B}}{\mathbf{C}; \mathbf{A} <: \mathbf{C}; \mathbf{B}}}$$

$$\overset{(\text{ OwnNil })}{\mathbf{A}^{\circ} <: \mathbf{0}} \qquad \overset{(\text{ OwnProj })}{\mathbf{A}^{\circ} <: \mathbf{A}} \qquad \overset{(\text{ OwnOwn })}{\mathbf{A}^{\circ} <: \mathbf{A}^{\circ\circ}} \qquad \overset{(\text{ NilOwn })}{\mathbf{0} <: \mathbf{0}^{\circ}}$$

$$\overset{(\text{ OwnParSeq })}{\mathbf{A}^{\circ}; \mathbf{B} <: \mathbf{A}^{\circ} \mid \mathbf{B}} \qquad \overset{(\text{ OwnCong })}{\frac{\mathbf{A} <: \mathbf{B}}{\mathbf{A}^{\circ} <: \mathbf{B}^{\circ}}} \qquad \overset{(\text{ ParOwn })}{\mathbf{A}^{\circ} \mid \mathbf{B}^{\circ} <:> (\mathbf{A} \mid \mathbf{B})^{\circ}}$$

$$\eta : \texttt{rec}\ \alpha.U <:> \eta : U\{^{\alpha}/_{\texttt{rec}\ \alpha.U}\} \qquad \frac{\eta : U <: \eta : V}{\eta : \texttt{rec}\ \alpha.U <: \eta : \texttt{rec}\ \alpha.V}$$

Fig. 4. Subtyping Rules

Our type system is based on the following forms of formal judgments:

$\mathbf{A} <: \mathbf{B}$	(Subtyping)	$P :: \mathbf{A} \rhd \mathbf{B}$	(Networks)
$[M; t] :: \mathbf{A} \shortmid \sigma \rhd \mathbf{B} \shortmid \delta\, [U]$	(Objects)	$e :: \mathbf{A} \shortmid \sigma \rhd \mathbf{B} \shortmid \delta\, [U]$	(Expressions)

Subtyping judgments are interpreted as expected. For networks, the typing judgment assigns to the network P an "assume-guarantee" assertion of the form $\mathbf{A} \rhd \mathbf{B}$, cf. the adjunct of the composition operator of spatial logics [11]. Intuitively, a judgment $P :: \mathbf{A} \rhd \mathbf{B}$ asserts that if P is composed with any network Q that satisfies the typing $\mathbf{A}$, one is guaranteed to obtain a network $(P \mid Q)$ that satisfies the typing $\mathbf{B}$. This form of judgment is essential for achieving compositionality in our type system. In an expression typing judgment, e is the expression to be typed, $\mathbf{A}$ and $\mathbf{B}$ are typing environments, and U is a type. The auxiliary type environments σ and δ (and $\mathbf{A}$ and $\mathbf{B}$ as well) keep information about effects on the local state of objects, and will be further explained below (notice the $\shortmid$ symbol separating the global environments $\mathbf{A}$ and $\mathbf{B}$ from the state environments σ and δ in judgments, not to be confused with the $\mid$ type constructor).

What does it really mean for a network to satisfy a typing? As discussed above, types are semantically interpreted as properties (sets of networks) expressible in a spatial logic. In Section 3.1 below we will present in detail a

$$\frac{}{\texttt{nil} :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{A} \shortmid \sigma\,[\mathbf{0}]}\;(\text{TNil}) \qquad \frac{}{v :: v : T^{\circ} \shortmid \sigma \;\triangleright\; v : \mathbf{0} \shortmid \sigma\,[T]}\;(\text{TValue})$$

$$\frac{}{a!(v) :: v : T^{\circ} \shortmid \sigma, a : \mathbf{0} \;\triangleright\; \shortmid \sigma, a : T\,[\mathbf{0}]}\;(\text{TWrite}) \qquad \frac{}{a? :: \shortmid \sigma, a : T \;\triangleright\; \shortmid \sigma, a : \mathbf{0}\,[T]}\;(\text{TRead})$$

$$\frac{}{v.\texttt{l}(u) :: v : \texttt{l}(U)V \mid u : U \shortmid \sigma \;\triangleright\; \shortmid \sigma\,[V]}\;(\text{TCall}) \qquad \frac{[M; \mathbf{0}] :: \mathbf{A}^{\circ} \shortmid \overline{a : \mathbf{0}} \;\triangleright\; \shortmid\,[T]}{\texttt{new}[\overline{a}; M] :: \mathbf{A}^{\circ} \shortmid \;\triangleright\; \shortmid\,[T^{\circ}]}\;(\text{TNew})$$

$$\frac{\mathbf{A} \mathrel{<:} \mathbf{A}' \quad \mathbf{B}' \mathrel{<:} \mathbf{B} \quad V' \mathrel{<:} V \qquad e :: \mathbf{A}' \shortmid \sigma \;\triangleright\; \mathbf{B}' \shortmid \delta\,[V']}{e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[V]}\;(\text{TSub}) \qquad \frac{e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[U] \qquad e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[V]}{e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[U \wedge V]}\;(\text{TAnd})$$

$$\frac{e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[V]}{e :: \mathbf{A} \mid \mathbf{C} \shortmid \sigma, \phi \;\triangleright\; \mathbf{B} \mid \mathbf{C} \shortmid \delta, \phi\,[V]}\;(\text{TPar}) \qquad \frac{e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[V]}{e :: \mathbf{A}; \mathbf{C} \shortmid \sigma \;\triangleright\; \mathbf{B}; \mathbf{C} \shortmid \delta\,[V]}\;(\text{TSeq})$$

$$\frac{e_i :: \mathbf{B}_i \shortmid \sigma_i \;\triangleright\; \shortmid \delta_i\,[V_i] \qquad f :: \mathbf{C}, \overline{x : V^{\circ}} \shortmid \overline{\delta} \;\triangleright\; \mathbf{E}, \overline{x : \mathbf{0}} \shortmid \phi\,[U]}{\texttt{let}\ \overline{x = e}\ \texttt{in} f :: \Pi(\overline{\mathbf{B}}); \mathbf{C} \shortmid \overline{\sigma} \;\triangleright\; \mathbf{E} \shortmid \phi\,[U]}\;(\text{TLet})$$

Fig. 5. Typing Rules (Expressions).

logical semantics of types, around which our soundness proofs are developed. For now, we present our type system as a formal system, and explain from an intuitive perspective the various rules and the main results. Our type system is composed by four sets of rules, to derive judgments of the four forms listed above. In Fig. 4 we present the subtyping axioms and rules. Subtyping principles are motivated by selected natural properties of types, and reflect valid semantic entailments in our logic (cf. Proposition 3.7). A first set of rules states that $(- \mid -)$ and $\mathbf{0}$ define a commutative monoid. The rule $(\mathbf{A}; \mathbf{B}) \mid (\mathbf{C}; \mathbf{D}) \mathrel{<:} (\mathbf{A} \mid \mathbf{C}); (\mathbf{B} \mid \mathbf{D})$ expresses the basic interaction principle between sequential and independent composition, allowing us to derive, *e.g.*, $\mathbf{A} \mid \mathbf{B} \mathrel{<:} \mathbf{A}; \mathbf{B}$ (interleaving). The rules for $(-)^{\circ}$ are quite interesting, notice that $(-)^{\circ}$ and $(- \mid -)$ reveal a familiar algebraic structure. Not so familiar is the rule $\mathbf{A}^{\circ}; \mathbf{B} \mathrel{<:} \mathbf{A}^{\circ} \mid \mathbf{B}$, asserting a key principle involving sequential composition and ownership: the owned usage $\mathbf{A}^{\circ}$ is not active (yet), and thus $\mathbf{B}$ cannot causally depend on it. A further set of rules express congruence principles, and unfolding of recursion.

In Fig. 5, we present the typing rules for expressions. Intuitively, a expression typing judgment $e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[U]$ means that e, when in the presence of services conforming to $\mathbf{A}$ and in a store conforming to σ will, after termination, yield a value of type U, while leaving a store conforming to δ, and the given services in a state where they may be still used as specified by the residual typing $\mathbf{B}$. Notice that typing of expressions depends on typing of objects,

$$\frac{M \equiv (N \mid \mathtt{l}(x) = e) \qquad e :: \mathbf{A}, x : U \shortmid \sigma \;\triangleright\; \mathbf{B}, x : \mathbf{0} \shortmid \delta\,[V]}{[M;\mathbf{0}] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[\mathtt{l}(U)V]}\;(\text{TOCall}) \qquad \frac{}{[M;\mathbf{0}] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{A} \shortmid \sigma\,[\mathbf{0}]}\;(\text{TONil})$$

$$\frac{[M;t] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[U] \qquad [M;\mathbf{0}] :: \mathbf{B} \shortmid \delta \;\triangleright\; \mathbf{C} \shortmid \phi\,[V]}{[M;t] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{C} \shortmid \phi\,[U;V]}\;(\text{TOSeq})$$

$$\frac{[M;t] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[U] \qquad [N;u] :: \mathbf{C} \shortmid \sigma' \;\triangleright\; \mathbf{D} \shortmid \delta'\,[V]}{[M;t] :: \mathbf{A} \mid \mathbf{C} \shortmid \sigma,\sigma' \;\triangleright\; \mathbf{B} \mid \mathbf{D} \shortmid \delta,\delta'\,[U \mid V]}\;(\text{TOPar}) \qquad \frac{[M;\mathbf{0}] :: \mathbf{A}^\circ \shortmid \sigma \;\triangleright\; \shortmid \delta\,[T]}{[M;\mathbf{0}] :: \mathbf{A}^\circ \shortmid \sigma \;\triangleright\; \shortmid \delta\,[T^\circ]}\;(\text{TOOwn})$$

Fig. 6. Typing Rules (Objects).

$$\frac{}{\mathbf{0} :: \mathbf{A} \;\triangleright\; \mathbf{A}}\;(\text{TVoid}) \qquad \frac{P :: \mathbf{A} \;\triangleright\; \mathbf{B} \quad P \equiv Q}{Q :: \mathbf{A} \;\triangleright\; \mathbf{B}}\;(\text{TStruc}) \qquad \frac{[M;t] :: \mathbf{A} \shortmid \overline{s_i : V_i} \;\triangleright\; \mathbf{B} \shortmid \delta\,[T]}{n[M;\overline{s_i\langle n_i\rangle};t] :: \mathbf{A} \mid \Pi(n_i : V_i^\circ) \;\triangleright\; n : T}\;(\text{TObj})$$

$$\frac{P :: \mathbf{A} \;\triangleright\; \mathbf{B} \quad Q :: \mathbf{C} \;\triangleright\; \mathbf{D} \quad P \| Q}{P \mid Q :: \mathbf{A} \mid \mathbf{C} \;\triangleright\; \mathbf{B} \mid \mathbf{D}}\;(\text{TPar}) \qquad \frac{P :: \mathbf{A} \;\triangleright\; \mathbf{B} \quad Q :: \mathbf{B} \;\triangleright\; \mathbf{C} \quad P \| Q}{P \mid Q :: \mathbf{A} \;\triangleright\; \mathbf{C}}\;(\text{TComp})$$

$$\frac{P :: \mathbf{A} \;\triangleright\; \mathbf{B} \quad n \# \mathbf{A},\mathbf{B}}{(\boldsymbol{\nu} n)P :: \mathbf{A} \;\triangleright\; \mathbf{B}}\;(\text{TNew}) \qquad \frac{\mathbf{A} <: \mathbf{A}' \quad P :: \mathbf{A}' \;\triangleright\; \mathbf{B}' \quad \mathbf{B}' <: \mathbf{B}}{P :: \mathbf{A} \;\triangleright\; \mathbf{B}}\;(\text{TSub})$$

Fig. 7. Typing Rules (Networks).

through the rule for `new` $[M]$. To intuitively grasp the general meaning of typing rules for expressions rules, it is useful to keep in mind that in a judgment $e :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[U]$, the return type U, as well as the stored types σ, δ, are implicitly "owned" (*e.g.*, we avoid writing U° in the return type $[U]$). Consistently, in the rule for a value (name or variable), the value v may be returned only if its type is owned (T°). The same happens in the rule for a write $a!(v)$, where ownership of some T view of v is handed over from the thread to the store. Notice how read / write effects are recorded in the left (σ) and right (δ) environments. Typing a method call $v.\mathtt{l}(u)$ requires separation between the method server v and the argument u. However, it does not force u, v to be different objects: non-interference is here ensured by the spatial typing via $- \mid -$, stating that the method part and the argument part do not share resources. We also have some congruence rules, a subtyping rule, and the rule for `let`. In the `let` rule, each expression e_i is required not to interfere with a concurrent e_j, by reading and writing in the local store. This condition will be slightly relaxed in Section 4, after the introduction of shared variables and types. Notice that the values returned from each e_i, whose evaluation depends on separate resources $\mathbf{B}_i$, are separate owned values, each of type V_i°.

In Fig. 7, we present the typing rules for objects and networks. Intuitively, the judgment $[M \parallel t] :: \mathbf{A} \mathrel{\text{I}} \sigma \;\triangleright\; \mathbf{B} \mathrel{\text{I}} \delta\,[U]$ asserts that any object $n[M \parallel s \parallel t]$ whose store s satisfies σ, upon composition with any network satisfying $\mathbf{A}$ can be safely used according to type the U. The residuals $\mathbf{B}$ and δ reflect the state of the external and local resources after the usage specified by U has been completed. Under this intuitive reading, we believe that all the rules for objects are already quite transparent, and the same remark also applies to the rules for networks. We discuss a bit the rule for object introduction. The rule requires that all state elements are distinctly named, and that each of the stored values n_i must be available in the environment for ownership by the object (as specified by $V_i{}^{\circ}$). Although in a perhaps subtle way, subtyping plays a key role in the derivation of expression, objects and network judgments, as the factorization of a substantial amount of structural reasoning in the subtyping relation contributed to keep our typing rules reasonably clean (we omit the obvious rule for subtyping object judgments).

Example 2.2 (continued). We now assign types to the system components. For F and H we may expect the typings $F :: \;\triangleright f : T_f$ and $H :: \;\triangleright h : T_h$, where

$$T_f \triangleq \mathtt{rec}\ \alpha.\mathtt{flight}(); (\mathtt{book}() \wedge \mathtt{free}()); \alpha$$
$$T_h \triangleq \mathtt{rec}\ \alpha.\mathtt{hotel}(); (\mathtt{book}() \wedge \mathtt{free}()); \alpha$$

For the gateway G, we let $G :: bk : T_{bank} \;\triangleright\; gw : T_{gw}$ where

$$T_{bk} \triangleq \mathtt{rec}\ \alpha.\mathtt{debit}()\mathtt{bool}; \alpha \qquad\qquad T_{gw} \triangleq \mathtt{rec}\ \alpha.\mathtt{pay}(\mathtt{book}() \wedge \mathtt{free}()); \alpha$$

Set $T_{br} \triangleq \mathtt{rec}\ \alpha.(\mathtt{flight}() \mid \mathtt{hotel}()); \mathtt{order}(); \alpha$. Now, the following judgment is derivable: $(F \mid H \mid G \mid B) :: bk : T_{bk} \;\triangleright\; br : T_{br}$.

This judgment asserts that the network $(F \mid H \mid G \mid B)$, when composed with any system providing the T_{bk} type at bk, will be safe for use at br as specified by type T_{br}. Such typing may be obtained compositionally from the types of the subsystems in many ways. A possible root level split of the system may be between the broker $B :: gw : T_{gw}, f : T_f, h : T_h \;\triangleright\; br : T_{br}$ and the back-end subsystem $(G \mid H \mid F) :: bk : T_{bk} \;\triangleright\; gw : T_{gw}, f : T_f, h : T_h$, where we conclude by the forward composition rule.

We define the following variant of the Kleene iterator: $T^{\otimes} \triangleq \mathtt{rec}\ \alpha.(T; \alpha)^{\circ}$. Notice that we have $T^{\otimes}$ <:> $(T; T^{\otimes})^{\circ}$ <: $\mathbf{0} \wedge (T; T^{\otimes})$. Hence, $T^{\otimes}$ can be unfolded infinitely many times into copies of T (as T^* does), but also be stored and returned by method calls, since it is an owned type (while T^* may not).

Example 2.11 (continued). For the spooler S, we propose the following typings. First, we abbreviate $T_{res} \triangleq (\mathtt{use}())^{\otimes}$, $T_{rm} \triangleq \mathtt{rec}\ \alpha.\mathtt{alloc}()T_{res}; \mathtt{free}(T_{res}{}^{\circ}); \alpha$ and $T_{srv} \triangleq \mathtt{rec}\ \alpha.\mathtt{open}(); T_{res}; \mathtt{close}(); \alpha$. Then the following is derivable: $S :: pool : T_{rm} \;\triangleright\; server : T_{srv}$. Notice how owned types ($T_{res}{}^{\circ}$) are used *to express ownership transfer of resources from the pool to the spooler and back. In general, we would expect a resource pool such as the one expected at pool to be shared by multiple users, while here the T_{rm} type just captures a very particular sequential usage. We will return to this point in Section 4 below.*

The safety properties ensured by our type system may be formally expressed in many ways. The fundamental consequence of typing is stuck-freeness, from which, as discussed in Section 2, other properties follow, such as race absence for unshareable resources, and conformance to usage protocols. We can thus already hint to our main soundness result, in a somewhat specific form.

Claim. Let $P :: \ \triangleright\ n : \mathtt{l}(\mathbf{0})\mathbf{0}$. Then there is Q such that $P \overset{n.\mathtt{l}_c(\mathtt{nil})}{\longrightarrow} Q$ and for all R such that $Q \Rightarrow R$ it is not the case that stuck(R).

The claim states that any network typed by $n : \mathtt{l}(\mathbf{0})\mathbf{0}$ offers a method $\mathtt{l}$ at object n that, after invoked, is ensured to evolve into a stuck state. More general safety results follow as direct consequence of the semantics of types, as developed in the next section, refining the preliminary presentation in [6].

3.1 Logical Semantics of Types

Any typing environment $\mathbf{A}$ denotes a certain property $[\![\mathbf{A}]\!]$, in the sense that if P is assigned type $\mathbf{A}$, then soundness of our type system ensures that $P \in [\![\mathbf{A}]\!]$, or, in terms of logical satisfaction, that $P \models \mathbf{A}$. In fact, we will not interpret types as properties directly, but will rather embed types in a more primitive spatial logic, so that each typing environment $\mathbf{A}$ is interpreted by a certain formula $\mathcal{A}$. The satisfaction predicate $\models$ is inductively defined on the structure of formulas, in such a way that $P \models \mathcal{A}$ implies that P enjoys certain general safety properties, in particular, stuck-freeness.

Definition 3.2 (Logic) *The set $\mathcal{F}$ of formulas is defined by:*

$$\begin{aligned} \mathcal{A}, \mathcal{B} ::= {} & \mathcal{A} \wedge \mathcal{B} \mid \forall x.\mathcal{A} \mid \mathcal{A} \mid \mathcal{B} \mid \mathcal{A} \triangleright \mathcal{B} \\ \mid {} & \mathbf{0} \mid \mathcal{A};\mathcal{B} \mid \mathcal{A}^{\circ} \mid (\boldsymbol{\nu})\mathcal{A} \mid n : \mathtt{l}_c(m) \mid n : c(\mathcal{A})V \end{aligned}$$

As in [7, 8, 5], our logic includes (positive) first-order logic, the basic spatial operators of composition and its adjunct with their standard meanings, and certain specific operators, in particular some behavioral modalities. Instead of including action prefixing modalities, we introduce a general sequential composition formula of the form $\mathcal{A};\mathcal{B}$, where $\mathcal{A}$ is interpreted both as a property, and as a *usage pattern*. Usage patterns are modeled by *usage*, a transition relation between networks and labeled by formulas, noted $P \overset{\mathcal{A}}{\longmapsto} Q$. The intuitive meaning of $P \overset{\mathcal{A}}{\longmapsto} Q$ is that if P is used (by the environment) as specified by

$$
\begin{array}{lll}
P \models \mathcal{A} \wedge \mathcal{B} & \textit{iff} & P \models \mathcal{A} \textit{ and } P \models \mathcal{B} \\
P \models \mathcal{A} \mid \mathcal{B} & \textit{iff} & \textit{exists } Q, R.\ P \equiv Q \mid R \textit{ and } Q \models \mathcal{A} \textit{ and } R \models \mathcal{B} \\
P \models \mathcal{A} \triangleright \mathcal{B} & \textit{iff} & \textit{forall } Q.\ \textit{if } (P \| Q) \textit{ and } Q \models \mathcal{A} \textit{ then } P \mid Q \models \mathcal{B} \\
P \models \forall x.\mathcal{A} & \textit{iff} & \textit{forall } n.\ P \models \mathcal{A}\{^x/_n\} \\
P \models \mathbf{0} & \textit{iff} & \mathrm{idle}(P) \\
P \models \mathcal{A}^\circ & \textit{iff} & P \models \mathcal{A} \textit{ and } P \models \mathbf{0} \\
P \models \mathcal{A}; \mathcal{B} & \textit{iff} & P \models \mathcal{A} \textit{ and forall } Q.\ \textit{if } P \xmapsto{\mathcal{A}} Q \textit{ then } Q \models \mathcal{B} \\
P \models n : \mathtt{1}_c(m) & \textit{iff} & \textit{exists } Q.\ \mathrm{idle}(P) \textit{ and } P \xrightarrow{n.\mathtt{1}_c(m)} Q \\
P \models (\boldsymbol{\nu})\mathcal{A} & \textit{iff} & \textit{exists } Q.\ P \equiv (\boldsymbol{\nu}\overline{m})Q \textit{ and } Q \models \mathcal{A} \textit{ and } \overline{m} \# \mathit{fn}(\mathcal{A}) \\
P \models n : c(\mathcal{A})V & \textit{iff} & \textit{forall } R, Q.\ \textit{if } (P \| R) \textit{ and } R \models \mathcal{A} \textit{ and } P \mid R \Rightarrow Q \\
 & & \textit{then } \neg\mathrm{stuck}(Q) \textit{ and} \\
 & & \quad \textit{forall } Q', r.\ \textit{if } Q \xrightarrow{\overline{n.c(r)}} Q' \textit{ then} \\
 & & \qquad \textit{exists } P', R', R_v.\ Q' \equiv P' \mid R' \mid R_v \textit{ and} \\
 & & \qquad\quad R_v \models r : V^\circ \textit{ and } R \xmapsto{\mathcal{A}} R'
\end{array}
$$

Fig. 8. Satisfaction

$$
P \xmapsto{\mathbf{0}} P \qquad
\frac{P \xmapsto{\mathcal{U}} Q}{P \xmapsto{\mathcal{U} \wedge \mathcal{V}} Q} \qquad
\frac{P \xmapsto{\mathcal{V}} Q}{P \xmapsto{\mathcal{U} \wedge \mathcal{V}} Q} \qquad
\frac{P \xmapsto{\mathcal{U}\{^x/_n\}} Q}{P \xmapsto{\forall x.\mathcal{U}} Q} \qquad
\frac{P \xrightarrow{n.\mathtt{1}_c(m)} Q}{P \xmapsto{n:\mathtt{1}_c(m)} Q}
$$

$$
\frac{P \equiv (\boldsymbol{\nu}\overline{m})R \quad R \xmapsto{\mathcal{U}} Q}{P \xmapsto{(\boldsymbol{\nu})\mathcal{U}} Q} \qquad
\frac{P \xmapsto{\mathcal{U}} R \quad R \xmapsto{\mathcal{V}} Q}{P \xmapsto{\mathcal{U};\mathcal{V}} Q} \qquad
P \xmapsto{\mathcal{U}^\circ} (P \setminus \mathcal{U}^\circ)
$$

$$
\frac{R \models \mathcal{A} \quad P \mid R \Rightarrow \xrightarrow{\overline{n.c(r)}} Q \quad R \xmapsto{\mathcal{A}} R'}{P \xmapsto{n:c(\mathcal{A})V} Q \setminus R' \setminus r : V^\circ}
$$

$$
\frac{P \equiv P_1 \mid P_2 \quad P_1 \xmapsto{\mathcal{U}} Q_1 \quad P_2 \xmapsto{\mathcal{V}} Q_2 \quad Q_1 \mid Q_2 \equiv Q}{P \xmapsto{\mathcal{U} \mid \mathcal{V}} Q}
$$

Fig. 9. Usage

$\mathcal{A}$, it may evolve to Q. Satisfaction and usage are defined by mutual induction.

Definition 3.3 (Satisfaction) Satisfaction*, noted* $P \models \mathcal{A}$*, and* typed usage*, noted* $P \xmapsto{\mathcal{A}} Q$ *are defined in Fig. 9. We also define* $[\![A]\!] \triangleq \{P \mid P \models A\}$.

To avoid clashes between fresh names introduced in the subsidiary transitions, the rule for $P \xmapsto{\mathcal{U} \mid \mathcal{V}} Q$ is subject to the provisos $Q_1 \| Q_2$ and $(\mathit{fn}(Q_1) \setminus \mathit{fn}(P_1)) \# (\mathit{fn}(Q_2) \setminus \mathit{fn}(P_2))$. The semantics of $n : \mathtt{1}_c(m)$ and $n : c(\mathcal{A})V$ is defined from the labeled transition system and external actions (Definition 2.7). Intuitively, P satisfies $n : c(\mathcal{A})V$ if P contains a thread c that whenever passed a resource R satisfying $\mathcal{A}$, is guaranteed to evolve in a stuck free way until a value r is returned, while exercising on R an usage specified by $\mathcal{A}$. Thus, the $c(\mathcal{A})V$ operator enforces both safety and liveness properties. The definition of usage for most type constructors is not unexpected. The usage for U° is particularly interesting: we have $P \overset{U^\circ}{\longmapsto} Q$ where $Q = P \setminus U^\circ$, where $P \setminus A$ is the complement of property A w.r.t. the process P.

Definition 3.4 (Complement) *For any formula A, we denote by $(P \setminus A)$ the $\leqq$-largest Q such that $P \equiv R \mid Q$ and $R \models A$. For any process R, we denote by $P \setminus R$ the $\leqq$-largest Q such that $P \equiv R \mid Q$.*

The complement $(P \setminus A)$ of P w.r.t. A, when it exists, it is unique (proof in appendix, Lemma A.2). The usage for U° models the situation where a minimal "piece" of the system (in the sense of $\leqq$, Definition 2.4) that satisfies property U is taken away (ownership is passed from the system to the environment). The usage for $n : c(\mathcal{A})V$ mimics the satisfaction clause, but notice how the returned resource $(r : V^\circ)$ and the residual R' of the argument is removed from the target. However, if the argument R is typed by an owned type (say, U°) the usage $R \overset{U^\circ}{\longmapsto} R'$ ensures that ownership is passed from the environment (caller) to the system. Using these ingredients, we introduce our interpretation of types. For any type environment $\mathbf{A}$, we define a formula $\lceil \mathbf{A} \rceil$:

Definition 3.5 *The embedding of types into logical formulas is given by:*

$$\begin{array}{ll@{\qquad}ll}
\lceil n : \mathbf{0} \rceil & \triangleq \mathbf{0} & \lceil n : U \mid V \rceil & \triangleq (\boldsymbol{\nu})(\lceil n : U \rceil \mid \lceil n : V \rceil) \\
\lceil n : U^\circ \rceil & \triangleq (\boldsymbol{\nu}) \lceil n : U \rceil^\circ & \lceil n : U ; V \rceil & \triangleq (\boldsymbol{\nu})(\lceil n : U \rceil ; \lceil n : V \rceil) \\
 & & \lceil n : U \wedge V \rceil & \triangleq \lceil n : U \rceil \wedge \lceil n : V \rceil \\
 & & \lceil \mathbf{A}, \mathbf{B} \rceil & \triangleq \lceil \mathbf{A} \rceil \mid \lceil \mathbf{B} \rceil \\
 & & \lceil n : \mathtt{l}(U)V \rceil & \triangleq \forall u, c \,.\, n : \mathtt{l}_c(u); n : c(u : U)V
\end{array}$$

Thus, all types are interpreted "directly", except method and thread types, which are interpreted in terms of finer grain primitives. Building on this interpretation, we define validity of subtyping and typing judgments as follows:

Definition 3.6 (Validity of typing and subtyping judgments)

$$\mathit{valid}(\mathbf{A} \mathrel{<:} \mathbf{B}) \triangleq [\![\lceil \mathbf{A} \rceil]\!] \subseteq [\![\lceil \mathbf{B} \rceil]\!] \qquad \mathit{valid}(P :: \mathbf{A} \rhd \mathbf{B}) \triangleq P \models \lceil \mathbf{A} \rceil \rhd \lceil \mathbf{B} \rceil$$

$$\begin{array}{l}
\mathit{valid}([M; t] :: \mathbf{A} \shortmid \sigma \rhd \mathbf{B} \shortmid \delta\, [T]) \triangleq \\
\quad \text{forall } n, \overline{n_i} \,.\, n[M \parallel \overline{s_i \langle n_i \rangle} \parallel t] \models \\
\qquad\qquad\qquad (\mathbf{A} \mid \Pi(n_i : \sigma(s_i)^\circ) \rhd (n : T); (\mathbf{B} \mid \Pi(n_i : \delta(s_i)^\circ))
\end{array}$$

$$\begin{array}{l}
\mathit{valid}(e :: \mathbf{A}, \overline{x : T} \shortmid \sigma \rhd \mathbf{B}, \overline{x : S} \shortmid \delta\, [V]) \triangleq \\
\quad \text{exists } \mathbf{C}, \mathbf{U} \,.\, \mathbf{A} \mathrel{<:} \mathbf{C}; \mathbf{B} \text{ and } \overline{x : T} \mathrel{<:} \overline{x : U; S} \,.\, \text{and forall } n, \overline{n_i}, \overline{p_k}. \\
\quad n[\ \parallel \overline{s_i \langle n_i \rangle} \parallel c \left\langle e \overline{\{{}^{x_k}\!/\!_{p_k}\}} \right\rangle] \models \\
\qquad\qquad\qquad \mathbf{A} \mid \Pi(n_i : \sigma(s_i)^\circ) \rhd c(\overline{p : U})V; (\mathbf{B} \mid \Pi(n_i : \delta(s_i)^\circ))
\end{array}$$

From now on, we will sometimes write typing environments where formulas are expected, having in mind the interpretation just presented. Our interpretation enjoys several nice properties. For example, the property stated in the *Claim* above (right before Section 3.1) is a direct consequence of the definition of the logical predicate $\models$. We can now state our main results (proofs in appendix):

Proposition 3.7 (Soundness of Subtyping) *If* $\mathbf{A} <: \mathbf{B}$ *then* $[\![\mathbf{A}]\!] \subseteq [\![\mathbf{B}]\!]$.

Theorem 3.8 (Soundness of Typing) *If* $P :: \mathbf{A} \triangleright \mathbf{B}$ *then* $P \models \mathbf{A} \triangleright \mathbf{B}$.

The proof technique we have developed here may be seen as an instance of the general method of logical relations, well understood in the setting of functional programming, but still quite unexplored in a concurrency setting. In a similar way, we build on a semantic interpretation of typed terms, which is defined by induction on types (as formulas), and then prove soundness by induction on typing derivations. Our result establishing validity under substitution for derivable expression-typing judgments then plays the role of the so-called Basic Lemma in the logical relations method. Because types are directly interpreted as properties of networks, our soundness results allows us to conclude:

Proposition 3.9 *Let* $P \models \mathbf{A}$ *and* $\mathbf{A} <: \mathbf{B}; \mathbf{C}$. *If* $P \overset{\mathbf{B}}{\longmapsto} Q$ *then* $Q \models \mathbf{C}$.

Proposition 3.9 is a semantic counterpart of the more familiar syntactic subject reduction property. In our case, it is an immediate consequence of the soundness of subtyping and of the semantics of $\mathbf{B}; \mathbf{C}$. By interpreting the semantic clause for the type $n : \mathtt{l}(U)V$, we also have:

Proposition 3.10 (Stuck Freeness) *Let* $P :: \ \triangleright n : \mathtt{l}(U)V$, *and* R *be such that* $R \models m : U$ *and* $R \| P$. *For all* Q *such that* $P \mid R \overset{n.\mathtt{l}_c(m)}{\longrightarrow}\Rightarrow Q$ *we have* $\neg\mathrm{stuck}(Q)$. *Moreover, if* $Q \xrightarrow{\overline{n.c(r)}} Q'$ *for some* Q', *then* $Q' \models r : V^\circ$.

A further consequence of Proposition 3.10, is that in well-typed processes all objects are used according to their intended usage protocols, as specified by the assigned spatial-behavioral types, and no races on method calls occur. More generally, Theorem 3.8 ensures that, after composition with an object BA such that $BA \triangleright bk : T_{bk}$, the system $(F \mid H \mid G \mid B)$ of Example 2.2 may be used according to the protocol T_{br}, without getting stuck, and ensuring that all objects H, G, F, etc, are used according to their intended protocols.

4 Resource Sharing and Shared Types

Although our framework already seems fairly powerful, it still prevents useful forms of sharing to be typable. While race absence may be a desirable correctness property of concurrent programming in general, in many situations, races are not problematic if the involved resources may be safely shared (*e.g.*, reading variables of "pure" type values, such as integers). Moreover, many common system resources are assumed to be intrinsically raceful (*e.g.*, semaphores, buffers). Sharing is also particularly useful to allow different threads to communicate by side effects. In this section, we show how sharing is accommodated in our framework. First, we add *shared store elements* ($*a\langle v\rangle$) and *shared methods* ($*\mathtt{l}(x) = e$) to the basic syntax of our calculus, together with structural congruence axioms to specify replication of shared methods:

$$\begin{array}{lll} M ::= \cdots \mid *\mathtt{l}(x) = e & \qquad *\mathtt{l}(x) = e \;\equiv\; (*\mathtt{l}(x) = e \mid *\,\mathtt{l}(x) = e) \\ s \;\;::= \cdots \mid *a\langle v\rangle & \qquad *\mathtt{l}(x) = e \;\equiv\; (*\mathtt{l}(x) = e \mid \quad \mathtt{l}(x) = e) \end{array}$$

The semantics defined in Section 2 is augmented by the following two rules:

$$\frac{e \xrightarrow{*a?(v)} e'}{n[\;\|\;\; *a\langle v\rangle \;\;\|\; c\langle e\rangle] \to n[\;\|\;\;\|\; c\langle e'\rangle]} \qquad \frac{e \xrightarrow{*a!(v)} e'}{n[\;\|\;\;\|\; c\langle e\rangle] \to n[\;\|\;\; *a\langle v\rangle \;\;\|\; c\langle e'\rangle]}$$

Thus, an object store may possibly record several values under the same shared tag $*a$, so that *e.g.*, $*a\langle 1\rangle \mid *a\langle 2\rangle$ is a valid store, although $a\langle 1\rangle \mid a\langle 2\rangle$ is not. Reading from the shared store consumes a state element, and writing to the shared store posts new state elements: the shared store behaves as a tuple space, that may be accessed concurrently by several threads within the same object. For technical convenience, we require that shared methods bodies do not reference other free variables than the parameter x: this does not bring any loss of generality, since any external reference may still be accessed through the local state. For processes to be composable, we require that public object slices agree on shared methods.

$$\begin{array}{lll} *\mu \in \mathit{smeth}(P, n) & \triangleq & P \equiv n[*\mu \;\|\;\;\|\;] \mid R \\ P \|_* Q & \triangleq & P \| Q \textit{ and for all } n.\ \mathit{smeth}(P, n) = \mathit{smeth}(Q, n) \end{array}$$

More interesting extensions relate to typing, the challenge is then to discipline shared access to the store of objects and of object slices. To that end, we separately assign types to the shared and unshared parts of the store. The intent is that while the types of the values stored under a given tag in the unshared part may dynamically change (cf. the spooler example 2.11), values stored under a given tag in the shared part must satisfy a fixed invariant. Since the shared part may suffer interference from parallel running threads, we rely on this invariant to ensure soundness (cf. the rely-guarantee approach [22]). To type shared usages of objects we introduce shared types, defined (by abbreviation) as $!U \triangleq \mathtt{rec}\ \alpha.(\mathbf{0} \wedge U \wedge (\alpha \mid \alpha))$. Shared types satisfy expected subtyping principles, namely $!U$ `<:>` $!U \mid !U$, $!U$ `<:` U and $!U$ `<:` $\mathbf{0}$. The first principle allows a service of type $!U$ to be used simultaneously by an unbounded number of clients. We may also derive $!U$ `<:` $U; !U$.

Example 4.1 *Consider the object* $NL \triangleq nl[\ *\texttt{null}() = \texttt{nil} \parallel\ \parallel]$. *It offers a method* `null` *that whenever called returns the value* `nil`. *Clearly, the service provided by NL may be shared by an arbitrary number of clients, without incurring in any execution error (stuck state) or protocol violation. So we expect the typing* $NL :: \ \triangleright\ !\texttt{null}()\mathbf{0}$.

Example 4.2 *Consider the following code for an object BF:*

$$BF \triangleq buf[\ *\texttt{put}(x) = *a!(x) \mid *\texttt{get}() = *a? \parallel\ \parallel]$$

BF implements a resource buffer. It keeps in its local shared state a bunch of references for resources of some type R°. *If all* $*a\,\langle - \rangle$ *cells are ensured to always hold values of type* R°, *we expect to obtain the typing* $BF :: \ \triangleright\ buf : !(\texttt{put}(R^\circ) \wedge \texttt{get}()R)$. *This type would allow multiple clients to share the buffer, while calling both methods, possibly concurrently. The typing of the* `put` *method demands ownership of the argument (via the type* R°*). We may formally specify the given shared store invariant by the assertion* $buf.a : R$. *An alternative typing for the same object is* $BF :: \ \triangleright\ buf : !\texttt{put}(R^\circ) \mid !\texttt{get}()R$. *This latter typing would allow BF to be used as an (unordered) queue, in a context where a bunch of writers use the* $buf : !\texttt{put}(R^\circ)$ *view, while a bunch of readers use the* $buf : !\texttt{get}()R$ *view of BF. Notice that although the methods* `put` *and* `get` *interfere through the local store of BF, according to our intended semantics the associated views are still safely separable by* $(- \mid -)$ *(up to any store manipulations that conform to the sharing invariant* $buf.a : R$*).*

We now describe the technical development necessary to accommodate sharing in the sense discussed above in our type system. Basically, we extend our basic typing judgments with a additional slot (ς, η), that specifies (by means of typing) the intended invariants on the shared stores. We thus introduce

$$P :: \mathbf{A} \mid \varsigma \ \triangleright\ \mathbf{B} \quad \text{(Networks)} \qquad [M;t] :: \mathbf{A} \mid \sigma \mid \eta \ \triangleright\ \mathbf{B} \mid \delta\,[U] \quad \text{(Objects)}$$
$$e :: \mathbf{A} \mid \sigma \mid \eta \ \triangleright\ \mathbf{B} \mid \delta\,[U] \quad \text{(Expressions)}$$

The sharing slot of object and expression typing judgments contains normal typing environments, while the sharing slot of network typing judgments contain a located typing environment. A *located typing environment* (ς) is a finite partial mapping from $\mathcal{N} \times \mathcal{N}$ to $\mathcal{T}$. We write $\varsigma \triangleq n_1.p_{n_{11}} : U_i, \ldots, n_k.p_n : U_{n_{kr_n}}$ for the located typing environment that assigns type U_i to $(n_i, p_{n_{ij}})$, that is, to the contents of shared state cells $p_{n_{ij}}$ of object n_i. We also write $\overline{n.p_i : U_i}$ as a shorthand for such a located typing environment, where the p_i range over the names of the shared state cells of each object n. We explain the general approach with a few typing rules in Fig. 10, the complete set of typing rules is presented in the appendix (Figs. B.1, B.2, and B.3).

The $\overline{n.p_i : U_i}$ (or $\overline{p_i : U_i}$) assertion in the sharing slot of the typing judgments specifies what are the admissible interferences from the environment, asserting

$$\overset{(\text{ TSWrite })}{*a? :: \;\shortmid \sigma \shortmid \delta, a : T \;\triangleright\; \shortmid \sigma\,[T]} \qquad \overset{(\text{ TSRead })}{*a!(v) :: v : T^{\circ} \shortmid \sigma \shortmid \delta, a : T \;\triangleright\; \shortmid \sigma\,[\mathbf{0}]}$$

$$\frac{[M;t] :: \mathbf{A} \shortmid \overline{s_i : V_i} \shortmid \overline{p_i : U_i} \;\triangleright\; \mathbf{B};\delta\,[T]}{n[M;\overline{s_i\langle v_i\rangle} \mid \overline{*p_{i_j}\left\langle r_{i_j}\right\rangle};t] :: \mathbf{A} \mid \Pi(v_i : V_i^{\circ}) \mid \Pi(r_{i_j} : U_i^{\circ}) \shortmid \overline{n.p_i : U_i} \;\triangleright\; n : T} \;(\text{ TSObj })$$

$$\frac{P :: \mathbf{A} \mid !\mathbf{S} \shortmid \varsigma \;\triangleright\; \mathbf{B} \quad Q :: \mathbf{C} \mid !\mathbf{S} \shortmid \varsigma \;\triangleright\; \mathbf{D} \quad P \|_{*} Q}{P \mid Q :: \mathbf{A} \mid \mathbf{C} \mid !\mathbf{S} \shortmid \varsigma \;\triangleright\; \mathbf{B} \mid \mathbf{D}} \;(\text{ TSPar })$$

Fig. 10. Some Sharing Typing Rules

$$\begin{array}{ll}
P \models_{\varsigma} \mathscr{A} \mid \mathscr{B} & \textit{iff exists } Q, R.\ P \equiv Q \mid R \textit{ and } Q \models_{\varsigma} \mathscr{A} \textit{ and } R \models_{\varsigma} \mathscr{B} \\
P \models_{\varsigma} n : c(\mathscr{A})V & \textit{iff forall } R, Q.\ \textit{if } (P \|_{*} R) \textit{ and } R \models_{\varsigma} \mathscr{A} \textit{ and } P \mid R \Rightarrow_{\varsigma} Q \\
& \quad \textit{then } \neg\mathrm{stuck}(Q) \textit{ and} \\
& \quad\; \textit{forall } Q', r.\ \textit{if } Q \xrightarrow{\overline{n.c(r)}} Q' \textit{ then} \\
& \quad\;\; \textit{exists } P', R', R_v.\ Q' \equiv P' \mid R' \mid R_v \textit{ and} \\
& \quad\;\;\; R_v \models r : V^{\circ} \textit{and } R \overset{\mathscr{A}}{\longmapsto}_{\varsigma} R'
\end{array}$$

Fig. 11. Sharing-indexed Satisfaction

that the store of each object n may only be modified (written or read) on its state elements $p_i\langle - \rangle$ if the invariant that such state elements will always contain values of type $U_i{}^{\circ}$ is preserved. The soundness of the sharing type system is proven as in Section 3.1, by semantical means. In the current setting, we introduce a sharing-indexed logical predicate $\models_{\varsigma}$, that takes into account possible interferences through the shared stores of objects as specified by the located typing environment ς. Thus $P \models_{\varsigma} \mathscr{A}$ means that P satisfies $\mathscr{A}$ in any computational context that reads from and writes to the stores of objects in P as specified by ς. We show a few clauses of the inductive definition of $\models_{\varsigma}$ in Fig. 11, the complete definition is presented in the appendix. The environment ς plays its essential role in the satisfaction clause for $n : c(\mathscr{A})V$; for the other logical operators, ς is compositionally propagated in the structure of formulas. Intuitively, $P \models_{\varsigma} n : c(\mathscr{A})V$ if P contains a thread c that whenever passed a resource R satisfying $\mathscr{A}$, is guaranteed to evolve in a stuck free way, subject to possible interferences through the objects' store as specified by ς, until a value r is returned, while exercising on R an usage specified by $\mathscr{A}$. Evolution (reduction) of a system subject to interferences as specified by a located typing environment ς is defined by interference-sensitive reduction, defined as follows:

Definition 4.3 (Interference-sensitive reduction) *Given a located typing environment ς, we define $\longrightarrow_{\varsigma}$ to be the least relation specified by:*

$$\frac{P \longrightarrow Q}{P \longrightarrow_{\varsigma} Q} \qquad \frac{\varsigma(n.a) = U}{P \mid n[\;\|\; a\langle v\rangle \;\|\;] \longrightarrow_{\varsigma} P \setminus v : U^{\circ}} \qquad \frac{R \models_{\varsigma} v : U^{\circ} \quad \varsigma(n.a) = U}{P \longrightarrow_{\varsigma} Q \mid R \mid n[\;\|\; a\langle v\rangle \;\|\;]}$$

Thus $\longrightarrow_{\varsigma}$ coincides with $\longrightarrow$ except that object state elements of the appropriate type (as specified by ς) may be read and written from / to the environment. We note $\Rightarrow_{\varsigma}$ the transitive reflexive closure of $\longrightarrow_{\varsigma}$. Validity of sharing typing judgments is defined as in Definition 3.6, but taking into account the sharing-index. We may then state and prove (details in the appendix).

Theorem 4.4 (Soundness of Typing) *If* $P :: \mathbf{A} \shortmid \varsigma \triangleright \mathbf{B}$ *then* $P \models_{\varsigma} \mathbf{A} \triangleright \mathbf{B}$.

We conclude the section showing how to type the spooler of our running example in a scenario with concurrent sharing and resource handover.

Example 2.11 (continued). Consider the implementation of a resource pool:

$$RP \triangleq pool[\, *\texttt{free}(x) = *a!(x) \mid *\texttt{alloc}() = *a? \parallel \ \parallel]$$

We may derive the typing $RP :: \ \shortmid pool.a : T_{res} \ \triangleright \ pool : T_p$ *where* $T_p \triangleq !(\texttt{free}(T_{res}{}^{\circ}) \wedge \texttt{alloc}() T_{res})$ *and* $T_{res} \triangleq (\texttt{use}())^{\otimes}$*. For the spooler S of Section 2 we may now derive* $S :: pool : T_p \shortmid \ \ \triangleright \ server : T_{\texttt{srv}}$*. Thus, we also have* $S :: pool : T_p \shortmid pool.a(T_{res}) \ \triangleright \ server : T_{\texttt{srv}}$*. Suppose we introduce another spooler* S' *where* $S' :: pool : T_p \shortmid pool.a : T_{res} \triangleright alt : T_{\texttt{srv}}$*. By* (TSPar)*, we have*

$$S \mid S' :: pool : T_p \shortmid pool.a : T_{res} \ \triangleright \ server : T_{\texttt{srv}} \mid alt : T_{\texttt{srv}}$$

We may now plug the resource pool RP in the "backend" of $S \mid S'$*, and get*

$$RP \mid S \mid S' :: \ \shortmid pool.a : T_{res} \ \triangleright \ server : T_{\texttt{srv}} \mid alt : T_{\texttt{srv}}$$

T so that the pool is shared by both spoolers. The resulting system is still open to interference, as specified by $pool.a : T_{res}$*: one may close it definitely by* (TSNew)*, to obtain* $(\boldsymbol{\nu} pool)(RP \mid S \mid S') :: \ \shortmid \ \ \triangleright \ server : T_{\texttt{srv}} \mid alt : T_{\texttt{srv}}$. Notice how owned types allow the resources (of type $T_{res}{}^{\circ}$) to be passed around dynamically from the pool to the spoolers and back, while fully respecting their usage protocols by the clients, in a scenario where the pool itself is shared.

5 Related Work and Discussion

The focus of this work is on a notion of spatial-behavioral typing, and its use to discipline interactions in concurrent distributed systems. To that end, we have presented a distributed object calculus inspired on well-established proposals [1, 3, 16]. We have adopted a distributed remote method invocation semantics, that involves a reply-answer protocol, and preserves the spatial distribution of threads in the system during method calls. This technical approach seems a rather natural choice, and a faithful account of actual implementations of distributed services. In our case, it was also imposed upon us by the necessity of expressing spatial decomposition of behaviors with respect to types, and then for our semantic soundness proofs to go through. Our type system enforces several safety properties on distributed object systems, in particular availability (meaning that method calls will be always served), and race absence with respect to unshareable resource access (meaning that the intended usage protocols of resources will be respected). Such properties result from the fact that our types are able to specify constraints on sequentiality of behavior, separation of resource access, and dynamic propagation of ownership, in a compositional way. Compositionality is certainly a desirable property of any verification method for concurrency, but it seems absolutely critical when one considers service-based systems, which are by nature open-ended, and dynamically assembled by relying on interface specifications or contracts. Thus, we would like to investigate how notions of types as developed here may be applied to other recently proposed service calculi, such as [4].

Our type system can be seen as a fragment of a spatial logic for concurrency [7, 8, 5], where the composition / separation operator plays a key role in ensuring resource control and non-interference. In our model, separation, up to structural congruence, cuts across the structure of objects, in order to separate safe computations involving both global and local resources. Spatial separation as a useful concept to reason about resource control and interference control in imperative programs was suggested in early works by Hoare [18] and Reynolds [29], and has recently motivated the development of the separation logics of O'Hearn and Reynolds [30, 25]. The work in this paper draws inspiration in some concepts introduced in that line of research, namely the use of the composition operator to talk about non-interference. It is then interesting to further discuss the relationship between the approaches.

Separation logics have been mainly used to talk about the heap structure, in a Hoare-style reasoning about imperative programs. In such models, actions are transformations on the passive heap state, while in our case states and actions are both represented by processes inside a common domain. Although our process model may be presented as an abstract separation algebra [10], in the case of dynamic spatial logic if an element satisfies $A \mid B$ this means that there are two independent subsystems that can safely accomplish the behaviors A and B, while in separation logic separation is used to reason about passive heap states, and behavior independently handled by Hoare triples. We find that the free combination of logical primitives to talk about spatial, or "intensional", aspects (such as resource usage or mobility) and behavioral aspects (such as the ability to perform actions) of processes in the same logic allows non-trivial properties of concurrent systems to be expressed in a rather uniform way. In any case, a dynamic spatial logic usually contains a separation logic for "processes-as-resources" as a fragment, notwithstanding its original motivation of reasoning about the structural dynamics of process systems by interpreting the process calculus static operators spatially.

In applications of separation logic to concurrency, simple imperative programs with conditional critical regions (CCRs) have been considered: in [26] it is discussed how separation logic assertions very conveniently specify how heap pieces may be safely passed around between threads, using a Hoare-style proof rule for CCRs [18]. In our case, types for the internal state of objects play a role similar to Hoare's resource invariants. However, we are able to deal with a richer computational model, in the sense that concurrent objects may be passed around as first-class entities, each such object carrying its own state and associated (possibly concurrently executing) operations. We represent resources as certain objects for which a certain concurrent usage protocol must be respected: this is a key ingredient of our approach. Spatial-behavioral logics turn out particularly convenient for reasoning about behavioral and resource-sensitive properties of such objects, of which, *e.g.*, shared memory cells are but a special case as discussed in the Introduction (cf. Strachey's load-update pairs). The "ownership hypothesis" as formulated by O'Hearn [26] also applies to our development, interpreted as ownership of references to interactive objects, rather than ownership of heap state. As a consequence, the ownership relation may be dynamically dictated by the (exported) type of the objects, rather than by assertions (using types or logic) on the (client) code that accesses shared entities, as in [26]. The notion of complement $(P \setminus A)$ (Definition 3.4) is related to the notion of footprint [10, 28], where the (minimal) part of the system P that satisfies A may be understood as the footprint of the behavior specified by A, even if here we talk of a formula footprint (which is a process), instead of a command footprint (which is a heap piece). Thus, fundamental concepts proposed in the context of the separation logics are also playing an important role in this work. A different way of exploiting separation logic within a process model (CSP-like, without name passing) was investigated by O'Hearn and Hoare [27], using a trace semantics, and building on an analogy between communication channels and heap cells.

A central claim of this work is that to reason about concurrent objects it is convenient to express in interfaces not only sequentiality constraints, as in more familiar behavioral or session type systems, but also *independence constraints*, stating what methods / operations may be simultaneously invoked, and when. We believe that such expressiveness is fairly important to support compositional reasoning about concurrent objects, although not previously investigated. We have introduced a owned type operator in our type structure. Our owned type constructor, aiming at the specification of dynamic ownership propagation, also seems to be new. A different notion of ownership type has been introduced in [15, 14], to enforce encapsulation in object-oriented programs, while we are introducing *owned types* to discipline the transfer of stateful objects between interacting processes, a fairly orthogonal concern.

The spatial interpretation of composition, together with owned types and compositional typing (via $\triangleright$), distinguishes our approach from other works on type systems for concurrent calculi that also combine composition and behavioral operations [20, 13]. In such works, parallel composition is interpreted behaviorally, as interleaving of actions, rather than as spatial separation of processes, as witnessed by several subtyping principles. It is out of the scope of such type systems to capture combined sequentiality / independence constraints, and to control transfer of ownership independently of name passing, as possible with owned types, *e.g.*, and illustrated in our broker and spooler examples. On the other hand, most works on types for concurrency have been applied to more fundamental models (variants of the π-calculus) for which spatial-behavioral typings as we are considering here may well turn out hard to develop. For example, in order to obtain more precise typings for sequential behaviors, in [23] a sequential composition operator was added to the π-calculus, the resulting language would be a possible subject to study combinations of classical types with spatial-behavioral types for the π-calculus. Some protocols definable in our type system are reminiscent of session types [19], it would be interesting to see how sessions might be representable in our setting. We also intend to investigate algorithmic presentations of our type system, in order to assess and demonstrate its practicality.

Unlike most works on type systems for concurrent calculi, we have adopted a semantic view of typing. The (original) understanding of types as properties has not always been a common guiding principle in the design of types for concurrent calculi, where a syntactical view seems to be dominant (however, see [12]). We find that a logic such as ours provides a suitable metalanguage in which many properties of interest may be formally expressed at an adequate level of abstraction, and that our proofs are much more intuitive and modular than more usual syntactic subject reduction style proof. For example, we are here able to deal with a very rich subtyping relation involving commutative monoidal operators in a straightforward modular way, while a syntactical proof would certainly have forced us to introduce technical artifacts, such as normal forms for types, and commutation results for derivations.

We have built on a logical relations (actually a unary logical predicate) technique to prove soundness. Although well-studied in the context of functional programming, logical relations have not been much explored in concurrency; an exception is [31], where they are applied to show termination of processes. It would then be interesting to understand these techniques in more general terms, a preliminary study along such directions was already presented in [9].

Acknowledgments. We would like to thank L. Cardelli, P. O'Hearn, V. Vasconcelos, E. Lozes, and G. Ferrari for related discussions. We acknowledge CITI, Project POSC/55582/2004 SpaceTimeTypes, and IP Sensoria.

References

[1] M. Abadi and L. Cardelli. A Theory of Primitive Objects: Untyped and First-Order Systems. *Information and Computation*, 125(2), 1996.

[2] M. Bartoletti, P. Degano, and G. Ferrari. Enforcing secure service composition. In *18th IEEE Computer Security Foundations Workshop*, pages 211–223. IEEE Computer Society, 2005.

[3] P. Di Blasio and K. Fisher. A Calculus for Concurrent Objects. In U. Montanari and V. Sassone, editors, *CONCUR '96, Concurrency Theory,*

7th International Conference, volume 1119 of *Lecture Notes in Computer Science*, pages 655–670. Springer-Verlag, 1996.

[4] M. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, and G. Zavattaro. SCC: a Service Centered Calculus. In *Proceedings of WS-FM 2006, 3rd International Workshop on Web Services and Formal Methods*, Lecture Notes in Computer Science. Springer-Verlag, 2006.

[5] L. Caires. Behavioral and Spatial Properties in a Logic for the Pi-Calculus. In I. Walukiewicz, editor, *Proc. of Foundations of Software Science and Computation Structures'2004*, number 2987 in Lecture Notes in Computer Science. Springer Verlag, 2004.

[6] L. Caires. Spatial-Behavioral Types, Distributed Services, and Resources. In U. Montanari and D. Sanella, editors, *TGC 2006 Second International Symposium on Trustworthy Global Computing*, Lecture Notes in Computer Science. Springer-Verlag, 2006.

[7] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Information and Computation*, 186(2):194–235, 2003.

[8] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). *Theoretical Computer Science*, 3(322):517–565, 2004.

[9] L. Caires. Logical Semantics of Types for Concurrency. In T. Mossakowski, U. Montanari, and M. Haveraaen, editors, *Algebra and Coalgebra in Computer Science, CALCO 2007*, volume 4624 of *Lecture Notes in Computer Science*, pages 16–35. Springer-Verlag, 2007.

[10] C. Calcagno, P. W. O'Hearn, and H. Yang. Local action and abstract separation logic. In *LICS 2007, 22nd IEEE Symposium on Logic in Computer Science*, pages 366–378. IEEE Computer Society, 2007.

[11] L. Cardelli and A. D. Gordon. Anytime, Anywhere. Modal Logics for Mobile Ambients. In *27th ACM Symp. on Principles of Programming Languages*, pages 365–377. ACM, 2000.

[12] G. Castagna, R. De Nicola, and D. Varacca. Semantic Subtyping for the π-Calculus. In *LICS 2005, 20th IEEE Symposium on Logic in Computer Science*, pages 92–101. IEEE Computer Society, 2005.

[13] S. Chaki, S. K. Rajamani, and J. Rehof. Types as Models: Model Checking Message-passing Programs. In *POPL 2002: The 29th Symposium on Principles of Programming Languages*, pages 45–57, 2002.

[14] D. G. Clarke and S. Drossopoulou. Ownership, encapsulation and the disjointness of type and effect. In *Proc. of the 2002 ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 292–310, 2002.

[15] D. G. Clarke, J. M. Potter, and J. Noble. Ownership types for flexible alias protection. In *Proceedings of the 13th Conference on Object-Oriented Programming, Systems, Languages, and Applications*, volume 33:10 of *ACM SIGPLAN Notices*, pages 48–64, New York, 1998. ACM Press.

[16] A. D. Gordon and P. D. Hankin. A Concurrent Object Calculus: Reduction and Typing. *Electr. Notes Theor. Comput. Sci.*, 16(3), 1998.

[17] M. Hennessy and J. Riely. Resource Access Control in Systems of Mobile Agents. *Information and Computation*, 173(1):82–120, 2002.

[18] C.A.R. Hoare. Towards a Theory of Parallel Programming. In C. Hoare and R. Perrot, editors, *Operating System Techniques.* Academic P., 1972.

[19] K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In C. Hankin, editor, *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 1998.

[20] A. Igarashi and N. Kobayashi. A Generic Type System for the Pi-Calculus. In *in POPL 2001: 28th Annual Symposium on Principles of Programming Languages*, 2001.

[21] A. Igarashi and N. Kobayashi. Resource Usage Analysis. In *POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 331–342, 2002.

[22] C. B. Jones. Specification and Design of (Parallel) Programs. In *IFIP Congress*, pages 321–332, 1983.

[23] U. de' Liguoro M. Dezani-Ciancaglini and N. Yoshida. On Progress for Structured Communications. In *Proceedings of the 3th Symposium on Trustworthy Global Computing TGC 2007*, 2007.

[24] R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[25] P. W. O'Hearn. Resources, Concurrency, and Local Reasoning. In P. Gardner and N. Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference*, volume 3170 of *Lecture Notes in Computer Science*, pages 49–67. Springer, 2004.

[26] P. W. O'Hearn. Resources, Concurrency, and Local Reasoning. *Theoretical Computer Science*, 375(1-3):271–307, 2007.

[27] P. W. O'Hearn. Separation Logic Semantics for Communicating Processes. In *Invited Lecture at Concur'07 - Unpublished*, 2007.

[28] M. Raza and P. Gardner. Footprints in Local Reasoning. In R. Amadio, editor, *Proc. of Foundations of Software Science and Computation Structures*, number 4962 in Lecture Notes in Computer Science. Springer Verlag, 2008.

[29] J. C. Reynolds. Syntactic control of interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, January 1978*, pages 39–46, 1978.

[30] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Third Annual Symposium on Logic in Computer Science*, Copenhagen, Denmark, 2002. IEEE Computer Society.

[31] D. Sangiorgi. Types, or: Where's the Difference Between CCS and pi? In L. Brim, P. Jancar, M. Kretínský, and A. Kucera, editors, *CONCUR 2002, 13th International Conference*, volume 2421 of *Lecture Notes in Computer Science*, pages 76–97. Springer-Verlag, 2002.

A Appendix to Section 3

Lemma A.1 *Some properties of satisfaction and usage.*

(1) Let $n \notin fn(A)$. Then $P \models A$ if and only if $(\boldsymbol{\nu} n)P \models A$.
(2) If $P \models A$ and $\mathrm{idle}(Q)$ then $P \mid Q \models A$.
(3) If $\mathrm{idle}(P)$ and $P \xmapsto{A} Q$ then $\mathrm{idle}(Q)$.
(4) Let $P_1 \models A$ and $P_2 \models B$ where $P_1 \| P_2$. If $P_1 \mid P_2 \xmapsto{A \mid B} Q$ then there are Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$, where $P_1 \xmapsto{A} Q_1$, and $P_2 \xmapsto{B} Q_2$.
(5) If $P \models A$ then there is Q such that $P \xmapsto{A} Q$.
(6) If $P \models A$ and $P \equiv Q$ then $Q \models A$.

Lemma A.2 *Some properties of complement.*

(1) Let $P \equiv Q_1 \mid R_1$ and $P \equiv Q_2 \mid R_2$, with $R_i \models A$ and $\leqq$-maximal for $i = 1, 2$, for some Q_1, Q_2, R_1, R_2 and Q_i. Then $R_1 \equiv R_2$.
(2) For all A, B and P, we have $P \setminus (A \mid B) = P \setminus A \setminus B$.
(3) For all A, B and P_1, P_2, we have $P_1 \setminus (A) \mid P_2 \setminus B = (P_1 \mid P_2) \setminus (A \mid B)$.

Proof. (1) Suppose $R_1 \not\equiv R_2$. Then $R_1 \equiv R_1' \mid R$, $R_2 \equiv R \mid R_2'$, where $R_1' \| R_2'$ and $R_1' \not\equiv \mathbf{0}$ or $R_2' \not\equiv \mathbf{0}$. We have $R_1' \mid R \models A$ and $R_2' \mid R \models A$. We may show, by induction on A, that $R \models A$ (intuitively, if R_1' is really needed in $R_1' \mid R$ to the satisfaction of A, then $R_2' \mid R$ has no way to emulate, given that $R_1' \| R_2'$). This would contradict the maximality of both R_1 and R_2. Hence $R_1 \equiv R_2$. ■

Proposition 3.7. If $\mathbf{A} <: \mathbf{B}$ then $[\![\mathbf{A}]\!] \subseteq [\![\mathbf{B}]\!]$.

Proof. By induction on the derivation of $\mathbf{A} <: \mathbf{B}$. For each subtyping rule, we show that if the premises are valid, so is the conclusion. Due to the role of usage in satisfaction, we need to show the following statement (2), that will be proven by simultaneous induction with the main statement (1):

(2) If $\mathbf{A} <: \mathbf{B}$, $P \models A$, $P \xmapsto{\mathbf{B}} Q$ then $P \xmapsto{\mathbf{A}} Q'$, for $Q \equiv Q' \mid R$ and $\mathrm{idle}(R)$.

We detail the proof of several interesting cases:

- (Case of (SeqAssocLR)) (1) Let $P \models (A; B); C$. Then $P \models A; B$ and for all R, if $P \xmapsto{A;B} R$ then $R \models C$. Then, $P \models A$ and for all Q, such that $P \xmapsto{A} Q$ then $Q \models B$. Also for all R such that $Q \xmapsto{B} R$, we have $R \models C$. Thus $Q \models B; C$. Hence $P \models A; (B; C)$. (2) Immediate.
- (Case of (SeqPar)) (1) Let $P \models (A; B) \mid (C; D)$. So $P \equiv (\boldsymbol{\nu}\overline{m})(P_1 \mid P_2)$ where $P_1 \models A; B$ and $P_2 \models C; D$. Also $P_1 \models A$ and for all Q' such that $P_1 \xmapsto{A} Q'$ we have $Q' \models B$. Likewise, $P_2 \models C$ and for all Q'' such that $P_2 \xmapsto{C} Q''$ we have $Q'' \models D$. We have $P \models A \mid C$. Pick R such that $P \xrightarrow{A \mid C}$

R. By Lemma A.1(4), there are R_1 and R_2 such that $R \equiv R_1 \mid R_2$ and $P_1 \overset{A}{\longmapsto} R_1$ and $P_2 \overset{C}{\longmapsto} R_2$. We conclude $R \models B \mid D$, so $P \models (A \mid C); (B \mid D)$. (2) Let $P \overset{(A \mid C);(B \mid D)}{\longmapsto} Q$, so that $P \overset{A \mid C}{\longmapsto} R \overset{B \mid D}{\longmapsto} Q$. This means that $P \equiv P_1 \mid P_2$ and $P_1 \overset{A}{\longmapsto} R_1$, $P_2 \overset{C}{\longmapsto} R_2$, $R \equiv R_1 \mid R_2 \equiv R_1' \mid R_2'$ and $R_1' \overset{B}{\longmapsto} Q_1'$, $R_2' \overset{D}{\longmapsto} Q_2'$ and $Q \equiv Q_1' \mid Q_2'$. Since $P \models (A; B) \mid (C; D)$, we also know that $P \models Q_1 \mid Q_2$ where $Q_1 \models A; B$ and $Q_2 \models C; D$. By Lemma A.1(4), we conclude $P \overset{A \mid C}{\longmapsto} R$, where $Q_1 \overset{A}{\longmapsto} R_1$ and $Q_2 \overset{C}{\longmapsto} R_2$. But then $R_1 \models B$ and $R_2 \models D$. By Lemma A.1(4), $R \overset{B \mid D}{\longmapsto} Q'$, where $R_1 \overset{B}{\longmapsto} Q_1''$, $R_2 \overset{D}{\longmapsto} Q_2''$ and $Q \equiv Q_1'' \mid Q_2''$. Since $Q_1 \overset{A;B}{\longmapsto} Q_1''$ and $Q_2 \overset{C;D}{\longmapsto} Q_2''$, we conclude $P \overset{(A;B) \mid (C;D)}{\longmapsto} Q$.

- (Case of (SeqCongL)) (1) Let $P \models A; C$. Then $P \models A$ and for all Q such that $P \overset{A}{\longmapsto} Q$ we have $P \models C$. By i.h., $P \models B$. Let $P \overset{B}{\longmapsto} Q$ for some Q. By i.h. (2), we have $P \overset{A}{\longmapsto} Q'$ where $Q \equiv Q' \mid R$ and idle(R). Since $Q' \models C$, by Lemma A.1(2) we conclude $Q \models C$. We conclude $P \models B; C$. (2) Let $P \overset{B;C}{\longmapsto} Q$. Then $P \overset{B}{\longmapsto} R \overset{C}{\longmapsto} Q$. By i.h. (2), $P \overset{A}{\longmapsto} R'$, where $R' \mid R'' \equiv R$ and idle(R''). We know that $R' \models C$. Hence $R' \overset{C}{\longmapsto} Q'$, where $Q \equiv Q' \mid R''$.
- (Case of (OwnParSeq)). (1) Let $P \models A^\circ; B$. Hence $P \models A^\circ$, and for all $Q = P \setminus A^\circ$ we have $Q \models B$. We conclude $P \models A^\circ \mid B$. (2) Let $P \overset{A^\circ \mid B}{\longmapsto} Q$. We have $P \equiv P_1 \mid P_2$ where $P_1 \overset{A^\circ}{\longmapsto} Q_1$ and $P_2 \overset{B}{\longmapsto} Q_2$. We have $P_1 \mid P_2 \overset{A^\circ \mid \mathbf{0}}{\longmapsto} Q_1 \mid P_2 \overset{\mathbf{0} \mid B}{\longmapsto} Q$, and conclude by Lemma A.1(4).
- (Case of (ParOwn)). (a) (1) Let $P \models A^\circ \mid B^\circ$. Hence $P \equiv (\boldsymbol{\nu} \overline{m})(P_1 \mid P_2)$ where $P_1 \models A$ and $P_1 \models A$ with idle(P_1) and idle(P_2). So idle(P) and $P_1 \mid P_2 \models A \mid B$, so $P \models (A \mid B)^\circ$. (2) Let $P \overset{(A \mid B)^\circ}{\longmapsto} Q$, so $Q = P \setminus (A \mid B)^\circ$. We have $P = P_1 \mid P_2$ where $P_1 \setminus A^\circ$ and $P_2 \setminus B^\circ$. Hence $P \overset{A^\circ \mid B^\circ}{\longmapsto} Q$.

  (b) (1) Let $P \models (A \mid B)^\circ$. So idle($P$) and $P \equiv (\boldsymbol{\nu} \overline{m})(P_1 \mid P_2)$ where $P_1 \models A$ and $P_1 \models A$. Then idle(P_1) and idle(P_2) so $P \models A^\circ \mid B^\circ$. (2) Let $P \overset{A^\circ \mid B^\circ}{\longmapsto} Q$, so $P = P_1 \mid P_2$ where $Q = (P_1 \setminus A^\circ) \mid (P_2 \setminus B^\circ)$. So $Q = P \setminus (A \mid B)^\circ$. We conclude $P \overset{(A \mid B)^\circ}{\longmapsto} Q$. ■

The proof of our next main result needs some build up and auxiliary Lemmas. We attempt to be both succint and clear, in order to present the complete proof in a reasonable amount of space.

Definition A.3 (Active context) *An active context is a syntactic type context where the hole appears in unguarded position. Active contexts are defined:*

$$\mathcal{E} ::= \square \mid \mathcal{E}; T \mid \mathcal{E} \mid T \mid T \mid \mathcal{E} \mid \mathcal{E}^\circ \mid \texttt{rec}\ \alpha.\mathcal{E}$$

Given a context $\mathcal{E}$ and a type T, we write $\mathcal{E}[T]$ for the type obtained by replacing the hole of $\mathcal{E}$ with T. We extend our definition of active contexts to typing environments and multi-hole contexts in the expected way.

Lemma A.4 *Let* $e :: \mathbf{A} \shortmid \sigma \ \triangleright\ \mathbf{B} \shortmid \delta\,[V]$*. Then valid*$(e :: \mathbf{A} \shortmid \sigma \ \triangleright\ \mathbf{B} \shortmid \delta\,[V])$*.*

Proof. By induction on the typing derivation, we prove the stronger property: Let $e :: \mathbf{A}, \overline{x : T} \shortmid \sigma \ \triangleright\ \mathbf{B}, \overline{x : S} \shortmid \delta\,[V]$. Then, for any active type context $\mathcal{F}[-]$,

$$\textit{exists } \mathbf{C}, \overline{U}\ .\ \mathbf{A} \mathrel{\texttt{<:}} \mathbf{C}; \mathbf{B}\ .\ \overline{x : T} \mathrel{\texttt{<:}} \overline{x : U; S}\ .\textit{for all } n, \overline{n_i}, \overline{m_k}.$$
$$n[\ \|\ \overline{s_i\,\langle n_i\rangle}\ \|\ c\left\langle e\overline{\{{}^{x_k}/_{m_k}\}}\right\rangle] \models \mathcal{F}[\mathbf{A}]\ \mid S_\sigma \ \triangleright\ c(\overline{m : U})V; (\mathcal{F}[\mathbf{B}]\ \mid S_\delta)$$

We write $P \mapsto\!\!\!\Rightarrow Q$ to refer to an arbitrary *stuck-free* reduction sequence from P to Q. We use the notation S_σ, etc, to refer to a formula of the form $\Pi_{n_i \in \mathfrak{D}(\sigma)}(n_i : \sigma(n_i)^\circ)$. We prove the result for a single substitution $\{{}^x/_m\}$, the general case is similar. When considering transition sequences, and decompositions up to $\mid$, we implicitly remove topmost name restrictions: by Lemma A.1(2), this does not invalidate our reasoning. We sometimes refer to a subtyping rule such as (NilOwn), to mean the corresponding semantic inclusion (by Lemma 3.7). We show the most interesting cases:

- (Case of (TValue)) We have $e :: \mathbf{A}' \shortmid \sigma \ \triangleright\ \mathbf{B}' \shortmid \delta\,[V]$ where $\mathbf{A}' = \mathbf{A}, x : T$ and $B' = \mathbf{A}, x : \mathbf{0}$ where $T = V^\circ$, $\mathbf{A} = \mathbf{B} = S = \mathbf{0}$, $\sigma = \delta$, and $e = v$.
  Set $N_o \triangleq n[\ \|\ \overline{s_i\,\langle n_i\rangle}\ \|\ c\,\langle m\rangle]$. Consider subcases (a) $e = x$ and (b) $e \neq x$.
  Case (a) Pick $P \models \mathcal{F}[\mathbf{A}] \mid S_\sigma$ and $R \models m : V^\circ$. Let
  $$P \mid R \mid N_o \mapsto\!\!\!\Rightarrow Q \xrightarrow{n.c(v)} Q'$$
  We must have $Q = P \mid R \mid N_o$ and $Q' = P \mid R \mid n[\ \|\ \overline{s_i\,\langle n_i\rangle}\ \|\ ]$ and $v = m$. We have $P \mid R \models \mathcal{F}[\mathbf{B}] \mid S_\delta \mid m : V^\circ$. Set $\mathbf{C} = \mathbf{0}$ and $U = V^\circ$.
  Thus $R \xmapsto{m:U} R'$ for some R'. We can check that $N_o \xmapsto{n:c(m:V^\circ)V} N'$ implies $N' \models \mathcal{F}[\mathbf{B}] \mid S_\delta$ (notice the only possible usage $R \xmapsto{m:V^\circ} R \setminus (m : V^\circ)$).
  So $N_o \models \mathcal{F}[\mathbf{A} \mid m : T] \mid S_\sigma \ \triangleright\ c(m : U)V; (\mathcal{F}[\mathbf{B}]\ \mid S_\delta)$.
  Case (b) is similar, except that we have $e = x = v$.
- (Case of (TNew)) We have $e :: \mathbf{A}' \shortmid \sigma \ \triangleright\ \mathbf{B}' \shortmid \delta\,[V]$ derived from $[M; \mathbf{0}] :: \mathbf{A}''^\circ, x : T'^\circ \ \triangleright\ \mathbf{0}\,[V]$, where $\mathbf{A}' = \mathbf{A}''^\circ, x : T$, and $\mathbf{B}' = \mathbf{0}$, and $T = T'^\circ$, and $\mathbf{A} = \mathbf{A}''^\circ$, and $\mathbf{B} = S = \mathbf{0}$ and $e = \texttt{new}\,[N]$, and $\sigma = \delta = \mathbf{0}$.
  Let $P \models \mathcal{F}[\mathbf{A}]$. By (ParSeq) and (OwnParSeq), $P \models m : T'^\circ \mid \mathbf{A}''^\circ \mid \mathcal{F}[\mathbf{0}]$. Thus $P \equiv (\boldsymbol{\nu}\overline{p})(P_1 \mid P_2)$ where $P_1 \models \mathcal{F}[\mathbf{0}]$ and $P_2 \models \mathbf{A}''^\circ$.
  Let $N_o \triangleq n[\ \|\ \ \|\ c\,\langle \texttt{new}[N]\rangle]$. Set $U \triangleq T = T'^\circ$. Pick $R \models m : T'^\circ$ and let $P \mid R \mid N_o \mapsto\!\!\!\Rightarrow Q \xrightarrow{n.c(v)} Q'$. So $Q = P \mid R \mid (\boldsymbol{\nu}f)(n[\ \|\ \ \|\ c\,\langle f\rangle] \mid f[N\ \|\ \ \|\ ])$ and $Q' = P \mid R \mid n[\ \|\ \ \|\ ] \mid f[N\ \|\ \ \|\ ]$, and $v = f$ (f fresh).
  By i.h., Lemma A.5, $P_2 \mid R \mid f[N\ \|\ \ \|\ ] \models f : V^\circ$. So, $Q' \models \mathcal{F}[\mathbf{B}] \mid f : V^\circ$. Set $\mathbf{C} \triangleq \mathbf{A} = \mathbf{A}''^\circ$. We have $P \xmapsto{m:U} R'$. As above, if $N_o \xmapsto{n:c(m:U)V} N'$ then $N' \models \mathcal{F}[\mathbf{B}]$. So $N_o \models \mathcal{F}[\mathbf{A}] \ \triangleright\ n : c(m : U)V; \mathcal{F}[\mathbf{B}]$.
- (Case of (TCall)) Let $e = v.\texttt{l}(u)$ and $e :: \mathbf{A}' \shortmid \sigma \ \triangleright\ \mathbf{B}' \shortmid \delta\,[V]$ where $\mathbf{A}' = v : \texttt{l}(E)V \mid u : E$, $\mathbf{B}' = \mathbf{0}$, and $\sigma = \delta$.
  For $(x = u = v)$, we have $e\{{}^x/_m\} = m.\texttt{l}(m)$, $\mathbf{A} = \mathbf{0}$, $\mathbf{B} = \mathbf{0}$, $S = \mathbf{0}$ and $T = \texttt{l}(E)V \mid E$. Pick $P \models \mathcal{F}[\mathbf{A}] \mid S_\sigma$. Set $N_o \triangleq n[M\ \|\ s\ \|\ c\,\langle m.\texttt{l}(m)\rangle]$, and pick $R \models m : \texttt{l}(E)V \mid E$. We first show that all transition sequences

of the form $R \xrightarrow{m.\mathtt{1}_d(m)} \Mapsto \xrightarrow{\overline{m.d(r)}} R'$ are such that $R' \equiv R'_1 \mid R'_2 \mid R_v$ where $R_v \models r : V^\circ$, and $R \equiv R_1 \mid R_2$ where $R_1 \xmapsto{m:\mathtt{1}(E)V} R'_1$ and $R_2 \xmapsto{m:E} R'_2$. Let $P \mid R \mid N_o \Mapsto Q \xrightarrow{\overline{n.c(v)}} Q'$. This sequence must have the form:

$$P \mid R \mid N_o \to P \mid n[\ \|\ s \ \|\ c\,\langle m.d()\rangle] \Mapsto$$

$$P \mid R' \mid n[\ \|\ s \ \|\ c\,\langle r\rangle] \xrightarrow{\overline{n.c(r)}} P \mid R' \mid n[\ \|\ s\ \|\ ]$$

Set $\mathbf{C} \triangleq \mathbf{0}$ and $U \triangleq E \mid \mathtt{1}(E)V$. We know that $R \xmapsto{m:U} R'$. Moreover $P \mid N_o \xmapsto{n:c(m:U)V} P \mid N_o$ and we notice that $P \mid N_o \models \mathcal{F}[\mathbf{A}] \mid S_\sigma$. Hence $N_o \models \mathcal{F}[\mathbf{A}] \mid S_\sigma \triangleright n : c(m : U)V; \mathcal{F}\,[\mathbf{B}] \mid S_\delta$.

For $(x = v, x \neq u)$, and thus $(u \neq v)$. We have $e\{{}^{x}/_{m}\} = m.\mathtt{1}(u)$, $\mathbf{A} = u : E$, $\mathbf{B} = \mathbf{0}$, $T = \mathtt{1}(E)V$, $S = \mathbf{0}$. Pick $P \models \mathcal{F}[u : E] \mid S_\sigma$. Hence $P \models (u : E); (\mathcal{F}[\mathbf{0}] \mid S_\sigma)$. Set $N_o \triangleq n[\ \|\ s\ \|\ c\,\langle m.\mathtt{1}(u)\rangle]$, and pick $R \models m : \mathtt{1}(E)V$.

We first show that all transition sequences $P \mid R \xrightarrow{m.\mathtt{1}_d(u)} \Mapsto \xrightarrow{\overline{m.d(r)}} Px$ are such that $Px \equiv P' \mid R' \mid R_v$ where $R_v \models r : V^\circ$, and $R \xmapsto{m:\mathtt{1}(E)V} R'$ and $P \xmapsto{m:E} P'$. Let $P \mid R \mid N_o \Mapsto Q \xrightarrow{\overline{n.c(v)}} Q'$. We conclude that this sequence must have the form:

$$P \mid R \mid N_o \to P \mid R \mid n[\ \|\ s\ \|\ c\,\langle m.d()\rangle] \Mapsto$$

$$Px \mid n[\ \|\ s\ \|\ c\,\langle r\rangle] \xrightarrow{\overline{n.c(r)}} Px \mid n[\ \|\ s\ \|\ ]$$

Set $\mathbf{C} \triangleq \mathbf{0}$ and $U \triangleq \mathtt{1}(E)V$. We know $R \xmapsto{m:U} R'$. Moreover $P \mid N_o \xmapsto{n:c(m:U)V} P' \mid N_o$. Since $P' \models (\mathcal{F}[\mathbf{0}] \mid S_\sigma)$, $N_o \models \mathcal{F}[\mathbf{A}] \mid S_\sigma \triangleright n : c(m : U)V; \mathcal{F}\,[\mathbf{B}] \mid S_\delta$.

$(x = u, x \neq v)$ As above, swapping the roles of u and v.

$(x \neq u, x \neq v)$ Also similar to the cases above.

- (Case of (TPar)) We have $e :: \mathbf{A} \mid \mathbf{D} \shortmid \sigma, \phi \triangleright \mathbf{B} \mid \mathbf{D} \shortmid \delta, \phi\,[V]$ concluded from $e :: \mathbf{A} \shortmid \sigma \triangleright \mathbf{B} \shortmid \delta\,[V]$. Let $\mathbf{A} \mid \mathbf{D} = (\mathbf{A}' \mid \mathbf{D}'), x : T_a \mid T_d$ and $\mathbf{B} \mid \mathbf{D} = (\mathbf{B}' \mid \mathbf{D}'), x : S_a \mid T_d$ and $\mathbf{A} = \mathbf{A}', x : T_a$ and $\mathbf{B} = \mathbf{B}', x : S_a$. Let $P \models \mathcal{F}[\mathbf{A}' \mid \mathbf{D}'] \mid S_\sigma \mid S_\phi$. By i.h. w.r.t the context $\mathcal{F}[\square \mid \mathbf{D}']$, we have $N_o \models \mathcal{F}[\mathbf{A}' \mid \mathbf{D}'] \mid S_\sigma \triangleright c(m : T'_a)V; (\mathcal{F}[\mathbf{B}'' \mid \mathbf{D}'] \mid S_\delta)$ where $\mathbf{A}' <: \mathbf{B}''; \mathbf{B}'$ and $T_a <: T'_a; S_a$. So $N_o \models \mathcal{F}[\mathbf{A}' \mid \mathbf{D}'] \mid S_{\sigma,\phi} \triangleright c(m : T'_a \mid T_d)V; (\mathcal{F}[\mathbf{B}'' \mid \mathbf{D}'] \mid S_{\delta,\phi})$, where $\mathbf{A}' \mid \mathbf{D}' <: \mathbf{B}''; (\mathbf{B}' \mid \mathbf{D}')$ and $T_a; T_d <: T'_a; (S_a \mid T_d)$.
- (Case of (TWrite)) We have $a!(v) :: v : V^\circ \shortmid \sigma, a : \mathbf{0} \triangleright \shortmid \sigma, a : V\,[\mathbf{0}]$. We consider two cases ($x = v$ and $x \neq v$).

  $(x = v)$. In this case, $\mathbf{A} = \mathbf{B} = \mathbf{C} = \mathbf{0}$, $T = U = V^\circ$, and $S = \mathbf{0}$. Let $N_o \triangleq n[\ \|\ \overline{s} \mid a\langle u\rangle\ \|\ c\,\langle a!(m)\rangle]$. Let $P \models \mathcal{F}[\mathbf{A}] \mid S_{\sigma,a:\mathbf{0}}$ and $R \models m : V^\circ$. Let $P \mid R \mid N_o \Mapsto Q' \xrightarrow{\overline{n.c(v)}} Q''$. This transition sequence must have the form

$$P \mid R \mid N_o \longrightarrow P \mid R \mid n[\ \|\ \overline{s}\ \|\ c\,\langle \mathtt{nil}\rangle] \xrightarrow{\overline{n.c(\mathtt{nil})}} P \mid R \mid n[\ \|\ \overline{s} \mid a\,\langle m\rangle\ \|\ ]$$

  where $P \mid R \models \mathcal{F}[\mathbf{A}] \mid S_{\sigma,a:V}$ and $R \xmapsto{m:U} R \setminus (m : V^\circ)$. We thus conclude $N_o \models \mathcal{F}[\mathbf{A}] \mid S_{\sigma,a:\mathbf{0}} \triangleright c(m : U)\mathbf{0}; (\mathcal{F}[\mathbf{B}] \mid S_{\sigma,a:V})$.

  $(x \neq v)$ In this case, $\mathbf{A} = v : V^\circ$, $\mathbf{C} = v : V^\circ$, $\mathbf{B} = \mathbf{0}$, $T = U = S = \mathbf{0}$. We consider $N_o \triangleq n[\ \|\ \overline{s}, a\langle u\rangle\ \|\ c\,\langle a!(v)\rangle]$. Let $P \models \mathcal{F}[\mathbf{A}] \mid S_{\sigma,a:\mathbf{0}}$ and $R \models \mathbf{0}$. Then $P \models \mathcal{F}[v : V^\circ] \mid S_{\sigma,a:\mathbf{0}}$ and then $P \models v : V^\circ \mid \mathcal{F}[\mathbf{0}] \mid S_{\sigma,a:\mathbf{0}}$. Let

$P \mid R \mid N_o \Mapsto Q' \xrightarrow{\overline{n.c(v)}} Q''$. This transition sequence must have the form

$$P \mid R \mid N_o \longrightarrow P \mid R \mid n[\,\| \overline{s} \| \, c\,\langle \mathtt{nil} \rangle] \xrightarrow{\overline{n.c(\mathtt{nil})}} P \mid R \mid n[\,\| \overline{s} \mid a\,\langle v \rangle \;\|\,]$$

where $P \models \mathcal{F}[\mathbf{0}] \mid S_{\sigma,a:V}$, $R \overset{\mathbf{0}}{\longmapsto} R$. We conclude $N_o \models \mathcal{F}[\mathbf{A}] \mid S_{\sigma,a:\mathbf{0}} \;\triangleright\; c(\mathbf{0})\mathbf{0}; (\mathcal{F}[\mathbf{B}] \mid S_{\sigma,a:V})$.

- (Case of (TLet)) We have $\mathtt{let}\; \overline{y = e}\; \mathtt{in} f :: \Pi(\overline{\mathbf{B}}); \mathbf{D} \shortmid \overline{\sigma} \;\triangleright\; \mathbf{H}\,[U]$ concluded from $e_i :: \mathbf{B}_i \shortmid \sigma_i \;\triangleright\; \mathbf{0} \shortmid \delta_i\,[V_i]$, for $i = 1 \ldots n$, and $f :: \mathbf{D}, \overline{y : V^\circ} \shortmid \overline{\delta} \;\triangleright\; \mathbf{H}, \overline{y : \mathbf{0}} \shortmid \phi\,[U]$.

  Assume $\Pi(\overline{\mathbf{B}}); \mathbf{D} = \Pi(\mathbf{B}'); \mathbf{D}', x : \Pi(\overline{E}); F$. Then $\mathbf{B}_i = \mathbf{B}'_i, x : \mathbf{E}_i$, and $T = \Pi(\overline{E}); F$. We also have $\mathbf{H} = \mathbf{H}', x : S$ and $\mathbf{D} = \mathbf{D}', x : F$. We consider here the case $n = 2$, the general case is similar. So, $(\mathbf{B}_1 \mid \mathbf{B}_2); \mathbf{D} = (\mathbf{B}'_1 \mid \mathbf{B}'_2); \mathbf{D}', x : (E_1 \mid E_2); F$ where $\mathbf{B}_i = \mathbf{B}'_i, x : E_i$, and $T = (E_1 \mid E_2); F$. We have $\mathbf{A} = (\mathbf{B}'_1 \mid \mathbf{B}'_2); \mathbf{D}'$.

  Let $N_{e_1} \triangleq n[\,\| \overline{s}_1 \| \, c\,\langle e_1\{^x/_m\}\rangle]$ and $N_{e_2} \triangleq n[\,\| \overline{s}_2 \| \, c\,\langle e_2\{^x/_m\}\rangle]$. By i.h. (for e_1), we have $N_{e_1} \models \mathbf{B}'_1; \mathbf{B}'_2; \mathcal{F}[\mathbf{D}'] \mid S_{\sigma_1} \;\triangleright\; c(m : E_1)V_1; (\mathbf{B}'_2; \mathcal{F}[\mathbf{D}'] \mid S_{\delta_1})$ and (for e_2) $N_{e_2} \models \mathbf{B}'_2; \mathbf{B}'_1; \mathcal{F}[\mathbf{D}'] \mid S_{\sigma_2} \triangleright c(m : E_2)V_2; (\mathbf{B}'_1; \mathcal{F}[\mathbf{D}'] \mid S_{\delta_2})$. Pick $P \models \mathcal{F}[\mathbf{A}]$ and $R \models m : T$. So $P \equiv (\boldsymbol{\nu}\overline{p})(P_1 \mid P_2)$, $P_1 \models \mathbf{B}'_1$ and $P_2 \models \mathbf{B}'_2$.

  Note that if $P_1 \mid R_1 \mid N_{e_1} \Mapsto Q_1 \xrightarrow{\overline{n.c(v_1)}} Q'_1$, by the property for N_{e_1}, we conclude $Q'_1 \equiv R_{v_1} \mid R'_1 \mid P'_1 \mid n[\,\| \overline{d_1} \|\,]$ where $R_{v_1} \models v_1 : V_1$, $R_1 \overset{m:E_1}{\longmapsto} R'_1$ and $P'_1 \models \mathbf{B}'_2; \mathcal{F}[\mathbf{D}'] \mid S_{\delta_1}$.

  Not that if $P_2 \mid R_2 \mid N_{e_2} \Mapsto Q_2 \xrightarrow{\overline{n.c(v_2)}} Q'_2$, by the property for N_{e_2}, we conclude $Q'_2 \equiv R_{v_2} \mid R'_2 \mid P'_2 \mid n[\,\| \overline{d_2} \|\,]$ where $R_{v_2} \models v_2 : V_2^\circ$, $R_2 \overset{m:E_2}{\longmapsto} R'_2$ and $P'_2 \models \mathbf{B}'_1; \mathcal{F}[\mathbf{D}'] \mid S_{\delta_2}$.

  Let $N_o \triangleq n[\,\| \overline{s_1, s_2} \| \, c\,\langle \mathtt{let}\; y_1 = e_1\{^x/_m\}, y_2 = e_2\{^x/_m\}\; \mathtt{in}\; f\{^x/_m\}\rangle]$. Given the independence of the transition sequences starting at $P_1 \mid R_1 \mid N_{e_1}$ and $P_1 \mid R_2 \mid N_{e_2}$, we conclude that all reductions $P \mid R \mid N_o \Mapsto Px \mid Nx$ where $Nx = n[\,\| \overline{d_1, d_2} \| \, c\,\langle \mathtt{let}\; y_1 = v_1, y_2 = v_2\; \mathtt{in}\; f\{^x/_m\}\rangle]$ must be such that $Px = R' \mid R_{v_1} \mid R_{v_2} \mid P'$ where $R' \models m : F$, $R_{v_i} \models v_i : V_i^\circ$, $P' \models \mathcal{F}[\mathbf{D}'] \mid S_{\delta_1,\delta_2}$, and $R \overset{m:E_1 \mid E_2}{\longmapsto} R'$.

  Let $N_f \triangleq n[\,\| \overline{d_1, d_2} \| \, c\,\langle f\{^x/_m\}\{^{y_1}/_{v_1}\}\{^{y_2}/_{v_2}\}\rangle]$. By i.h. (for f), we know that $N_f \models \mathcal{F}[\mathbf{D}'] \mid S_{\delta_1,\delta_2} \;\triangleright\; c(m : F \mid v_1 : V_1^\circ \mid v_2 : V_2^\circ)U; (\mathcal{F}[\mathbf{H}'] \mid S_\phi)$. But then, $N_f \mid R_v \models \mathcal{F}[\mathbf{D}'] \mid S_{\delta_1,\delta_2} \triangleright c(m : F)U; (\mathcal{F}[\mathbf{H}'] \mid S_\phi)$, and $N_f \mid R_v \mid P' \models c(m : F)U; (\mathcal{F}[\mathbf{H}'] \mid S_\phi)$. Therefore $P \mid R \mid N_o \Mapsto Px \mid Nx \longrightarrow \xrightarrow{\overline{n.c(r)}} P_f$, where for any such sequence we have $P_f \equiv P'' \mid R'' \mid V'' \mid n[\,\| \overline{a} \|\,]$ with $P'' \models \mathcal{F}[\mathbf{H}'] \mid S_\phi$ and $R \overset{m:T}{\longmapsto} R''$ and $V'' \models r : U^\circ$. We conclude $N_o \models \mathcal{F}[\mathbf{A}] \mid S_{\overline{\sigma}} \;\triangleright\; c(m : T)U; (\mathcal{F}[\mathbf{H}'] \mid S_\phi)$. ■

Lemma A.5 *Let the judgment* $[M \| s \| t] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[T]$ *be derivable. Then valid*$([M \| s \| t] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta\,[T])$.

Proof. By induction on the typing derivation.

- (Case of (TOnil)) We have derived $[M \| \overline{s_i\,\langle n_i \rangle} \| \mathbf{0}] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{A} \shortmid \sigma\,[\mathbf{0}]$. Let

$P \models \mathbf{A} \mid \Pi(n_i : \sigma(n_i)^\circ)$. Since $P \models \mathbf{A}$ implies $\mathtt{idle}(P)$ for any type A (not formula) and $\Pi(n_i : \sigma(n_i))^\circ \models \mathbf{0}$, we have that $P \models \mathbf{0}$. Thus $P \mid n[M \parallel s \parallel \mathbf{0}] \models \mathbf{0}$. If $P \mid n[M \parallel s \parallel \mathbf{0}] \xmapsto{\mathbf{0}} Q$ then $Q \equiv P \mid n[M \parallel s \parallel \mathbf{0}]$, hence $n[M \parallel s \parallel \mathbf{0}] \models \mathbf{A} \mid S_\sigma \,\triangleright\, n : \mathbf{0}; (\mathbf{A} \mid S_\sigma)$.

- (Case of (TOPar)) Let $[M_1 \mid M_2; t_1 \mid t_2] :: \mathbf{A} \mid \mathbf{C} \shortmid \sigma, \sigma' \triangleright \mathbf{B} \mid \mathbf{D} \shortmid \delta, \delta' [U \mid V]$ be concluded from $[M_1; t_1] :: \mathbf{A} \shortmid \sigma \triangleright \mathbf{B} \shortmid \delta[U]$ and $[M_2; t_2] :: \mathbf{C} \shortmid \sigma' \triangleright \mathbf{D} \shortmid \delta' [V]$. Pick $P \models \mathbf{A} \mid \mathbf{C} \mid S_\sigma \mid S_{\sigma'}$. Thus $P \equiv (\boldsymbol{\nu}\overline{p})(P_1 \mid P_2 \mid S_1 \mid S_2)$ where $P_1 \models \mathbf{A}$, $P_2 \models \mathbf{C}$, $S_1 \models S_\sigma$ and $S_2 \models S_{\sigma'}$. By i.h., $P_1 \mid S_1 \mid n[M_1 \parallel s_1 \parallel t_1] \models (n : U); (B \mid S_\delta)$ and $P_2 \mid S_2 \mid n[M_2 \parallel s_2 \parallel t_2] \models n : V; (D \mid S_{\delta'})$. So, $P_1 \mid P_2 \mid n[M_1 \mid M_2; s_1 \mid s_2; t_1 \mid t_2] \models n : U \mid V$. By (ParSeq)

$$P_1 \mid P_2 \mid n[M_1 \mid M_2; s_1 \mid s_2; t_1 \mid t_2] \models n : U \mid V; (\mathbf{B} \mid \mathbf{D} \mid S_\delta \mid S_{\delta'})$$

- (Case of (TOOwn)) We have $n[M; \mathbf{0}] :: \mathbf{A}^\circ \shortmid \;\triangleright\; \shortmid [T^\circ]$ concluded from $n[M; \mathbf{0}] :: \mathbf{A}^\circ \shortmid \;\triangleright\; \shortmid \mathbf{0}\, [T]$. Let $P \models \mathbf{A}^\circ$. By i.h., $P \mid n[M \parallel \; \parallel \mathbf{0}] \models n : T$, and for all R such that $P \mid n[M \parallel \; \parallel \mathbf{0}] \xmapsto{n:T} R$ we have $R \models \mathbf{0}$. Since $P \models \mathbf{0}$, we conclude $P \mid n[M \parallel \; \parallel \mathbf{0}] \models n : T^\circ$.
- (Case of (TOCall)) We have $[M; \mathbf{0}] :: \mathbf{A} \shortmid \sigma \;\triangleright\; \mathbf{B} \shortmid \delta[\mathtt{l}(T)V]$ concluded from $\mathtt{l}(x) = e \mid N \equiv M$ and $e : \mathbf{A}, x : T \shortmid \sigma \;\triangleright\; \mathbf{B}, x : \mathbf{0} \shortmid \delta[V]$. Pick $P \models \mathbf{A} \mid S_\sigma$ and some name m. Let $N_o \triangleq n[M \parallel s \parallel \mathbf{0}]$ and $S \triangleq P \mid N_o$. We have $N_o \xrightarrow{n.\mathtt{l}_c(m)} N$ where $N \triangleq n[M \parallel s \parallel c\,\langle e\{^x/_m\}\rangle]$.

  By Lemma A.4, there is $\mathbf{C}, U$ such that $\mathbf{A} \mathrel{<:} \mathbf{C}; \mathbf{B}$ and $T \mathrel{<:} U; \mathbf{0}$ (so that $T \mathrel{<:} U$) and $N \models \mathbf{A} \mid S_\sigma \;\triangleright\; n : c(m : U)V; (\mathbf{B} \mid S_\delta)$. Then $S \models \forall m \,.\, n : \mathtt{l}_c(m); c(m : U)V; (\mathbf{B} \mid S_\delta)$ and $S \models \forall m \,.\, n : \mathtt{l}_c(m); c(m : T)V; (\mathbf{B} \mid S_\delta)$. Therefore, $S \models n : \mathtt{l}(T)V; (\mathbf{B} \mid S_\delta)$, and since P was arbitrary, $N_o \models (\mathbf{A} \mid S_\sigma) \;\triangleright\; n : \mathtt{l}(T)V; (\mathbf{B} \mid S_\delta)$. ∎

Theorem 3.8. If $P :: \mathbf{A} \;\triangleright\; \mathbf{B}$ then $P \models \mathbf{A} \;\triangleright\; \mathbf{B}$.

Proof. By induction on the typing derivation.

- (Case of (TPar)) We have $P \mid Q :: \mathbf{A} \mid \mathbf{C} \triangleright \mathbf{B} \mid \mathbf{D}$ concluded from $P :: \mathbf{A} \triangleright \mathbf{B}$ and $Q :: \mathbf{C} \;\triangleright\; \mathbf{D}$. Let $R \models \mathbf{A} \mid \mathbf{C}$. Then $R \equiv (\boldsymbol{\nu}\overline{m})(R_1 \mid R_2)$ where $R_1 \models \mathbf{A}$ and $R_2 \models \mathbf{C}$. By i.h., we have that $P \mid R_1 \models \mathbf{B}$ and $Q \mid R_2 \models \mathbf{D}$. So, $R_1 \mid R_2 \mid P \mid Q \models \mathbf{B} \mid \mathbf{D}$. By Lemma A.1(1), $R \mid P \mid Q \models \mathbf{B} \mid \mathbf{D}$.
- (Case of (TComp)) We have $P \mid Q :: \mathbf{A} \;\triangleright\; \mathbf{C}$ conclude from $P :: \mathbf{A} \;\triangleright\; \mathbf{B}$ and $Q :: \mathbf{B} \;\triangleright\; \mathbf{C}$. Let $R \models \mathbf{A}$. By i.h., we have that $R \mid P \models \mathbf{B}$ and, again by i.h., $R \mid P \mid Q \models \mathbf{C}$.
- (Case of (TNew)) We have concluded $(\boldsymbol{\nu}n)P :: \mathbf{A} \;\triangleright\; \mathbf{B}$ from $P :: \mathbf{A} \;\triangleright\; \mathbf{B}$ where $n \# \mathbf{A}, \mathbf{B}$. Consider $(\boldsymbol{\nu}n)P$ and $R \models \mathbf{A}$ (we may assume $n \notin \mathit{fn}(R)$). By i.h., we have that $R \mid P \models \mathbf{B}$. By Lemma A.1(1), $(\boldsymbol{\nu}n)(R \mid P) \models \mathbf{B}$, and thus $R \mid (\boldsymbol{\nu}n)P \models \mathbf{B}$.
- (Case of (TSub)) By Proposition 3.7.
- (Case of (TObj)) Let $n[M \parallel \overline{s\langle n_i\rangle} \parallel t] :: \mathbf{A} \mid \Pi(n_i : V_i^\circ) \;\triangleright\; n : T$ be concluded from $[M; t] :: \mathbf{A} \shortmid \overline{s_i : V_i} : \;\triangleright\; \mathbf{B} \shortmid [T]$. By Lemma A.5, we have $n[M \parallel \overline{s\langle n_i\rangle} \parallel t] \models (\mathbf{A} \mid \Pi(n_i : V_i^\circ)) \;\triangleright\; (n : T); \mathbf{B}$. ∎

B Appendix to Section 4

In Figs. B.1, B.2, and B.3, we present the complete set of typing rules for the full type system with sharing. The rules are almost the same as for the basic system, with the addition of the sharing slot in judgments. The sharing slot plays its essential role in rules TSWrite and TSRead (expressions), TSPar and TSObjs (Networks), and is propagated in an additive fashion in the remaining typing rules. The soundness proof is developed along the same lines as in the basic case without sharing. We introduce the sharing-indexed logical predicate and the sharing-indexed usage, as defined in Figs B.4 and B.5. This allows us to define semantic validity for typing judgments.

Definition B.1 (Validity of sharing typing and subtyping judgments)

$$\mathit{valid}(P :: \mathbf{A} \shortmid \varsigma \ \triangleright\ \mathbf{B}) \triangleq P \models_{\varsigma} \mathbf{A} \ \triangleright\ \mathbf{B}$$

$$\begin{array}{l}
\mathit{valid}([M;t] :: \mathbf{A} \shortmid \sigma \shortmid \overline{p_i : U_i} \ \triangleright\ \mathbf{B} \shortmid \delta\,[T]) \triangleq \\
\quad \text{for all } n, \overline{n_i}, \overline{r_{i_j}} \ .\ n[M \parallel \overline{s_i\,\langle n_i\rangle} \mid \overline{*p_{i_j}\left\langle r_{i_j}\right\rangle} \parallel t] \models_{\overline{n.p_i:U_i}} \\
\qquad\qquad (\mathbf{A} \mid \Pi(n_i : \sigma(s_i)^{\circ}) \mid \Pi(r_{i_j} : U_i{}^{\circ}) \ \triangleright\ (n : T); (\mathbf{B} \mid \Pi(n_i : \delta(s_i)^{\circ}))
\end{array}$$

$$\begin{array}{l}
\mathit{valid}(e :: \mathbf{A}, \overline{x : T} \shortmid \sigma \shortmid \overline{p_i : U_i} \ \triangleright\ \mathbf{B}, \overline{x : S} \shortmid \delta\,[V]) \triangleq \\
\quad \text{exists } \mathbf{C}, \mathbf{U} \ .\ \mathbf{A} \mathrel{<:} \mathbf{C}; \mathbf{B} \text{ and } \overline{x : T} \mathrel{<:} \overline{x : U; S} \ .\ \text{and forall } n, \overline{n_i}, \overline{p_k}, \overline{r_{i_j}}. \\
\quad n[\ \parallel \overline{s_i\,\langle n_i\rangle} \mid \overline{*p_{i_j}\left\langle r_{i_j}\right\rangle} \parallel c\left\langle e\overline{\{{}^{x_k}/{}_{p_k}\}}\right\rangle] \models_{\overline{n.p_i:U_i}} \\
\qquad\qquad \mathbf{A} \mid \Pi(n_i : \sigma(s_i)^{\circ}) \mid \Pi(r_{i_j} : U_i{}^{\circ}) \ \triangleright\ c(\overline{p : U})V; (\mathbf{B} \mid \Pi(n_i : \delta(s_i)^{\circ}))
\end{array}$$

Again, soundness of the type system is proven by showing that each typing rule establishes validity of the judgment in its conclusion, given the validity of its premises. All properties in Lemma A.1, also carry to $\models_{\varsigma}$.

Lemma B.2 *Let* $e :: \mathbf{A} \shortmid \sigma \shortmid \eta \triangleright \mathbf{B} \shortmid \delta\,[V]$*. Then* $\mathit{valid}(e :: \mathbf{A} \shortmid \sigma \shortmid \eta \triangleright \mathbf{B} \shortmid \delta\,[V])$*.*

Proof. By induction on the typing derivation, we prove the stronger property. Let $e :: \mathbf{A}, \overline{x : T} \shortmid \sigma \shortmid \eta \ \triangleright\ \mathbf{B}, \overline{x : S} \shortmid \delta\,[V]$. For any active type context $\mathcal{F}\,[-]$,

$$\begin{array}{l}
\mathit{exists}\ \mathbf{C}, \mathbf{U} \ .\ \mathbf{A} \mathrel{<:} \mathbf{C}; \mathbf{B}\ \mathit{and}\ \overline{x : T} \mathrel{<:} \overline{x : U; S} \ .\ \mathit{and\ for\ all}\ n, \overline{n_i}, \overline{p_k}, \overline{r_{i_j}}. \\
\quad n[\ \parallel \overline{s_i\,\langle n_i\rangle} \mid \overline{*p_{i_j}\left\langle r_{i_j}\right\rangle} \parallel c\left\langle e\overline{\{{}^{x_k}/{}_{p_k}\}}\right\rangle] \models_{\overline{n.p_i:U_i}} \\
\qquad\qquad \mathcal{F}\,[\mathbf{A}] \mid S_{\sigma} \mid \Pi(r_{i_j} : U_i{}^{\circ}) \ \triangleright\ c(\overline{p : U})V; (\mathbf{B} \mid S_{\delta}))
\end{array}$$

- (Case of (TSRead)) We have $*a? :: \ \shortmid \sigma \shortmid \eta \ \triangleright\ \ \shortmid \sigma\,[V]$. Then $\mathbf{A} = \mathbf{B} = \mathbf{0}$, $T = S = \mathbf{0}$, and $\eta = \eta', a : V$. Let $\mathbf{C} = U = \mathbf{0}$. Let $N_o \triangleq n[\ \parallel \overline{s} \mid *\overline{p} \parallel c\,\langle a?\rangle]$, where $\varsigma = n.\eta$. Let $P \models_{\varsigma} \mathcal{F}[\mathbf{A}] \mid S_{\sigma} \mid S_{*p,\eta}$ and $R \models_{\varsigma} \mathbf{0}$.

  Let $P \mid R \mid N_o \Mapsto_{\varsigma} Q' \xrightarrow{\overline{n.c(v)}} Q''$. This sequence must have the form

$$\text{(TSNil)}\quad \texttt{nil} :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{A} \shortmid \sigma\, [\mathbf{0}]$$

$$\text{(TSValue)}\quad v :: v : T^{\circ} \shortmid \sigma \shortmid \eta \;\triangleright\; v : \mathbf{0} \shortmid \sigma\, [T]$$

$$\text{(TWrite)}\quad a!(v) :: v : T^{\circ} \shortmid \sigma, a : \mathbf{0} \shortmid \eta \;\triangleright\; \shortmid \sigma, a : T\, [\mathbf{0}]$$

$$\text{(TRead)}\quad a? :: \shortmid \sigma, a : T \shortmid \eta \;\triangleright\; \shortmid \sigma, a : \mathbf{0}\, [T]$$

$$\text{(TSWrite)}\quad *a!(v) :: v : T^{\circ} \shortmid \sigma \shortmid \eta, a : T \;\triangleright\; \shortmid \sigma\, [\mathbf{0}]$$

$$\text{(TSRead)}\quad *a? :: \shortmid \sigma \shortmid \eta, a : T \;\triangleright\; \shortmid \sigma\, [T]$$

$$\text{(TSCall)}\quad v.\mathtt{l}(u) :: v : \mathtt{l}(U)V \mid u : U \shortmid \sigma \shortmid \eta \;\triangleright\; \shortmid \sigma\, [V]$$

$$\text{(TSNew)}\quad \frac{[M; \mathbf{0}] :: \mathbf{A}^{\circ} \shortmid \overline{a : \mathbf{0}} \shortmid \eta \;\triangleright\; \shortmid\, [T]}{\texttt{new}[\overline{a}; M] :: \mathbf{A}^{\circ} \shortmid \;\shortmid \eta \;\triangleright\; \shortmid\, [T^{\circ}]}$$

$$\text{(TSSub)}\quad \frac{\begin{array}{c}\mathbf{A} \mathrel{<:} \mathbf{A}' \quad \mathbf{B}' \mathrel{<:} \mathbf{B} \quad V' \mathrel{<:} V \\ e :: \mathbf{A}' \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B}' \shortmid \delta\, [V']\end{array}}{e :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [V]}$$

$$\text{(TSAnd)}\quad \frac{\begin{array}{c}e :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [U] \\ e :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [V]\end{array}}{e :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [U \wedge V]}$$

$$\text{(TSPar)}\quad \frac{e :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [V]}{e :: \mathbf{A} \mid \mathbf{C} \shortmid \sigma, \phi \shortmid \eta \;\triangleright\; \mathbf{B} \mid \mathbf{C} \shortmid \delta, \phi\, [V]}$$

$$\text{(TSSeq)}\quad \frac{e :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [V]}{e :: \mathbf{A}; \mathbf{C} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B}; \mathbf{C} \shortmid \delta\, [V]}$$

$$\frac{\begin{array}{c}e_i :: \mathbf{B}_i \shortmid \sigma_i \shortmid \eta \;\triangleright\; \shortmid \delta_i\, [V_i] \\ f :: \mathbf{C}, \overline{x : V^{\circ}} \shortmid \overline{\delta} \shortmid \eta \;\triangleright\; \mathbf{E}, \overline{x : \mathbf{0}} \shortmid \phi\, [U]\end{array}}{\texttt{let}\ \overline{x = e}\ \texttt{in} f :: \Pi(\overline{\mathbf{B}}); \mathbf{C} \shortmid \overline{\sigma} \shortmid \eta \;\triangleright\; \mathbf{E} \shortmid \phi\, [U]}\quad \text{(TSLet)}$$

Fig. B.1. Sharing Typing Rules (Expressions).

$$\frac{\begin{array}{c}M \equiv (N \mid \mathtt{l}(x) = e) \quad \text{(TOSCall)} \\ e :: \mathbf{A}, x : U \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B}, x : \mathbf{0} \shortmid \delta\, [V]\end{array}}{[M; \mathbf{0}] :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [\mathtt{l}(U)V]}$$

$$\text{(TOSNil)}\quad [M; \mathbf{0}] :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{A} \shortmid \sigma\, [\mathbf{0}]$$

$$\frac{[M; t] :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [U] \quad [M; \mathbf{0}] :: \mathbf{B} \shortmid \delta \shortmid \eta \;\triangleright\; \mathbf{C} \shortmid \phi\, [V]}{[M; t] :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{C} \shortmid \phi\, [U; V]}\quad \text{(TOSSeq)}$$

$$\frac{\begin{array}{c}[M; t] :: \mathbf{A} \shortmid \sigma \shortmid \eta \;\triangleright\; \mathbf{B} \shortmid \delta\, [U] \quad \text{(TOSPar)} \\ [N; u] :: \mathbf{C} \shortmid \sigma' \shortmid \eta \;\triangleright\; \mathbf{D} \shortmid \delta'\, [V]\end{array}}{[M; t] :: \mathbf{A} \mid \mathbf{C} \shortmid \sigma, \sigma' \shortmid \eta \;\triangleright\; \mathbf{B} \mid \mathbf{D} \shortmid \delta, \delta'\, [U \mid V]}$$

$$\text{(TOSOwn)}\quad \frac{[M; \mathbf{0}] :: \mathbf{A}^{\circ} \shortmid \sigma \shortmid \eta \;\triangleright\; \shortmid \delta\, [T]}{[M; \mathbf{0}] :: \mathbf{A}^{\circ} \shortmid \sigma \shortmid \eta \;\triangleright\; \shortmid \delta\, [T^{\circ}]}$$

Fig. B.2. Sharing Typing Rules (Objects).

$$P \mid R \mid N_o \longrightarrow P \mid R \mid n[\ \| \ \overline{s} \mid \ * \overline{p'} \ \| \ c\,\langle v \rangle] \xrightarrow{\overline{n.c(v)}} P \mid R \mid n[\ \| \ \overline{s} \mid \ * \overline{p'} \ \| \ ]$$

where $*\overline{p'} = *\overline{p} \setminus *a\langle v \rangle$, $P \models_{\varsigma} \mathcal{F}[\mathbf{A}] \mid S_{\sigma} \mid v : V^{\circ}$ and $R \overset{\mathbf{0}}{\longmapsto} R$. We thus conclude $N_o \models_{\varsigma} \mathcal{F}[\mathbf{A}] \mid S_{\sigma} \mid S_{*p,\eta} \triangleright c(\mathbf{0}); (\mathcal{F}[\mathbf{A}] \mid S_{\sigma})$.

- (Case of (TSLet)) We have $\texttt{let}\ \overline{y = e}\ \texttt{in} f :: \Pi(\overline{\mathbf{B}}); \mathbf{D} \shortmid \overline{\sigma} \shortmid \varsigma \;\triangleright\; \mathbf{H}\, [U]$ concluded from $e_i :: \mathbf{B}_i \shortmid \sigma_i \shortmid \varsigma \;\triangleright\; \mathbf{0} \shortmid \delta_i\, [V_i]$, for $i = 1 \ldots n$, and $f :: \mathbf{D}, \overline{y : V^{\circ}} \shortmid \overline{\delta} \shortmid \varsigma \;\triangleright\; \mathbf{H}, \overline{y : \mathbf{0}} \shortmid \phi\, [U]$. We introduce the same abbreviations as

$$\overset{(\text{ TSVoid })}{\mathbf{0} :: \mathbf{A} \shortmid \varsigma \ \rhd \ \mathbf{A}} \qquad \overset{(\text{ TSStruc })}{\frac{P :: \mathbf{A} \shortmid \varsigma \ \rhd \ \mathbf{B} \quad P \equiv Q}{Q :: \mathbf{A} \shortmid \varsigma \ \rhd \ \mathbf{B}}}$$

$$\overset{(\text{ TSPar })}{\frac{\begin{array}{c} P :: \mathbf{A} \mid !\mathbf{S} \shortmid \varsigma \ \rhd \ \mathbf{B} \\ Q :: \mathbf{C} \mid !\mathbf{S} \shortmid \varsigma \ \rhd \ \mathbf{D} \ P \|_* Q \end{array}}{P \mid Q :: \mathbf{A} \mid \mathbf{C} \mid !\mathbf{S} \shortmid \varsigma \ \rhd \ \mathbf{B} \mid \mathbf{D}}} \qquad \overset{(\text{ TComp })}{\frac{\begin{array}{c} P :: \mathbf{A} \shortmid \varsigma \ \rhd \ \mathbf{B} \\ Q :: \mathbf{B} \shortmid \varsigma \ \rhd \ \mathbf{C} \ P \|_* Q \end{array}}{P \mid Q :: \mathbf{A} \shortmid \varsigma \ \rhd \ \mathbf{C}}}$$

$$\overset{(\text{ TSNew })}{\frac{P :: \mathbf{A} \shortmid \varsigma, \overline{n.p : T} \ \rhd \ \mathbf{B} \quad n \# \mathbf{A}, \mathbf{B}, \varsigma}{(\boldsymbol{\nu} n) P :: \mathbf{A} \shortmid \varsigma \ \rhd \ \mathbf{B}}} \qquad \overset{(\text{ TSSub })}{\frac{\mathbf{A} \mathrel{\texttt{<:}} \mathbf{A}' \quad P :: \mathbf{A}' \shortmid \varsigma \ \rhd \ \mathbf{B}' \quad \mathbf{B}' \mathrel{\texttt{<:}} \mathbf{B}}{P :: \mathbf{A} \shortmid \varsigma \ \rhd \ \mathbf{B}}}$$

$$\frac{[M; t] :: \mathbf{A} \shortmid \overline{s_i : V_i} \shortmid \overline{p_i : U_i} \ \rhd \ \mathbf{B}; \delta\, [T] \qquad (\text{ TSObj })}{n[M \parallel \overline{s_i \langle v_i \rangle} \mid \overline{*p_{i_j} \left\langle r_{i_j} \right\rangle} \parallel t] :: \mathbf{A} \mid \Pi(v_i : V_i^\circ) \mid \Pi(r_{i_j} : U_i^\circ) \shortmid \overline{n.p_i : U_i} \ \rhd \ n : T}$$

Fig. B.3. Sharing Typing Rules (Networks).

in the proof of Lemma A.4((TLet)). Let $N_{e_1} \triangleq n[\ \| \ \overline{s}_1 \mid * \overline{p}_1 \parallel c \langle e_1 \{{}^{x}\!/_{m}\} \rangle]$, $N_{e_2} \triangleq n[\ \| \ \overline{s}_2 \mid * \overline{p}_1 \parallel c \langle e_2 \{{}^{x}\!/_{m}\} \rangle]$.

By i.h., $N_{e_1} \models_\varsigma \mathbf{B}_1'; \mathbf{B}_2'; \mathcal{F}[\mathbf{D}'] \mid S_{\sigma_1} \mid S_{*p_1,\eta} \rhd c(m : E_1) V_1; (\mathbf{B}_2'; \mathcal{F}[\mathbf{D}'] \mid S_{\delta_1})$ and $N_{e_2} \models_\varsigma \mathbf{B}_2'; \mathbf{B}_1'; \mathcal{F}[\mathbf{D}'] \mid S_{\sigma_2} \mid S_{*p_2,\eta} \ \rhd \ c(m : E_2) V_2; (\mathbf{B}_1'; \mathcal{F}[\mathbf{D}'] \mid S_{\delta_2})$. Pick $P \models_\varsigma \mathcal{F}[\mathbf{A}] \mid S_{\overline{\sigma}}$ and $R_\varsigma \models m : T$. So $P \equiv (\boldsymbol{\nu} \overline{p})(P_1 \mid P_2)$, where $P_1 \models_\varsigma \mathbf{B}_1' \mid S_{\sigma_1}$ and $P_2 \models_\varsigma \mathbf{B}_2' \mid S_{\sigma_2}$.

Let $N_o \triangleq n[\ \| \ \overline{s_1, s_2} \ \| \ c \langle \texttt{let}\ y_1 = e_1 \{{}^{x}\!/_{m}\}, y_2 = e_2 \{{}^{x}\!/_{m}\}\ \texttt{in}\ f \{{}^{x}\!/_{m}\} \rangle]$. Given the conformance to ς of all reduction sequences starting at $P_1 \mid R_1 \mid N_{e_1}$ and $P_1 \mid R_2 \mid N_{e_2}$, and the properties stated above for N_{e_1} and N_{e_2}, we conclude that all reductions $P \mid R \mid N_o \mapsto\!\!\!\Rightarrow_\varsigma Px \mid Nx$ where

$$Nx = n[\ \| \ \overline{d_1, d_2} \parallel c \langle \texttt{let}\ y_1 = v_1, y_2 = v_2\ \texttt{in}\ f \{{}^{x}\!/_{m}\} \rangle]$$

must be such that $Px = R' \mid R_{v_1} \mid R_{v_2} \mid P'$ where $R' \models_\varsigma m : F$, $R_{v_i} \models v_i : V_i^\circ$, $P' \models_\varsigma \mathcal{F}[\mathbf{D}'] \mid S_{\delta_1,\delta_2}$, and $R \overset{m:E_1 \mid E_2}{\longmapsto} R'$. As in the proof of Lemma A.4 (TLet), we conclude $N_o \models_\varsigma \mathcal{F}[\mathbf{A}] \mid S_{\overline{\sigma}} \mid S_{*p,\eta} \ \rhd \ c(m : T) U; (\mathcal{F}[\mathbf{H}'] \mid S_\phi)$. ■

Lemma B.3 *Let the judgment* $[M \parallel s \parallel t] :: \mathbf{A} \shortmid \sigma \shortmid \eta \ \rhd \ \mathbf{B} \shortmid \delta\, [T]$ *be derivable. Then valid*$([M \parallel s \parallel t] :: \mathbf{A} \shortmid \sigma \shortmid \eta \ \rhd \ \mathbf{B} \shortmid \delta\, [T])$.

Proof. We show (Case of (TOSCall)). We have $[M; \mathbf{0}] :: \mathbf{A} \shortmid \sigma \shortmid \eta \rhd \mathbf{B} \shortmid \delta[\mathtt{l}(T) V]$ concluded from $\mathtt{l}(x) = e \mid N \equiv M$ and $e : \mathbf{A}, x : T \shortmid \sigma \shortmid \eta \ \rhd \ \mathbf{B}, x : \mathbf{0} \shortmid \delta[V]$. Pick $P \models \mathbf{A} \mid S_\sigma \mid S_{*p,\eta}$ and some name m. Let $N_o \triangleq n[M \parallel \overline{s} \mid * \overline{p} \parallel \mathbf{0}]$ and $Q \triangleq P \mid N_o$. We have $N_o \overset{n.\mathtt{l}_c(m)}{\longrightarrow} N$ where $N \triangleq n[M \parallel \overline{s} \mid * \overline{p} \parallel c \langle e \{{}^{x}\!/_{m}\} \rangle]$. By Lemma B.2, there is $\mathbf{C}, U$ such that $\mathbf{A} \mathrel{\texttt{<:}} \mathbf{C}; \mathbf{B}$ and $T \mathrel{\texttt{<:}} U; \mathbf{0}$ (so that $T \mathrel{\texttt{<:}} U$) and $N \models_\varsigma \mathbf{A} \mid S_\sigma \mid S_{*p,\eta} \ \rhd \ n : c(m : U) V; (\mathbf{B} \mid S_\delta)$. Then $Q \models_\varsigma \forall m \ .\ n : \mathtt{l}_c(m); c(m : U) V; (\mathbf{B} \mid S_\delta)$ and $Q \models_\varsigma \forall m \ .\ n : \mathtt{l}_c(m); c(m : T) V; (\mathbf{B} \mid S_\delta)$. Therefore, $Q \models_\varsigma n : \mathtt{l}(T) V; (\mathbf{B} \mid S_\delta)$, and since P was arbitrary, $N_o \models_\varsigma \mathbf{A} \mid S_\sigma \mid S_{*p,\eta} \ \rhd \ n : \mathtt{l}(T) V; (\mathbf{B} \mid S_\delta)$. ■

$$
\begin{array}{lll}
P \models_\varsigma \mathcal{A} \wedge \mathcal{B} & \textit{iff} & P \models_\varsigma \mathcal{A} \textit{ and } P \models_\varsigma \mathcal{B} \\
P \models_\varsigma \mathcal{A} \mid \mathcal{B} & \textit{iff} & \textit{exists } Q, R.\ P \equiv Q \mid R \textit{ and } Q \models_\varsigma \mathcal{A} \textit{ and } R \models_\varsigma \mathcal{B} \\
P \models_\varsigma \mathcal{A} \triangleright \mathcal{B} & \textit{iff} & \textit{forall } Q.\ \textit{if } (P \|_* Q) \textit{ and } Q \models_\varsigma \mathcal{A} \textit{ then } P \mid Q \models_\varsigma \mathcal{B} \\
P \models_\varsigma \forall x.\mathcal{A} & \textit{iff} & \textit{forall } n.\ P \models_\varsigma \mathcal{A}\{^x/_n\} \\
P \models_\varsigma \mathbf{0} & \textit{iff} & \mathrm{idle}(P) \\
P \models_\varsigma \mathcal{A}^\circ & \textit{iff} & P \models_\varsigma \mathcal{A} \textit{ and } P \models_\varsigma \mathbf{0} \\
P \models_\varsigma \mathcal{A};\mathcal{B} & \textit{iff} & P \models_\varsigma \mathcal{A} \textit{ and forall } Q.\ \textit{if } P \overset{\mathcal{A}}{\longmapsto}_\varsigma Q \textit{ then } Q \models_\varsigma \mathcal{B} \\
P \models_\varsigma n : \mathtt{1}_c(m) & \textit{iff} & \textit{exists } Q.\ \mathrm{idle}(P) \textit{ and } P \overset{n.\mathtt{1}_c(m)}{\longrightarrow} Q \\
P \models_\varsigma (\boldsymbol{\nu})\mathcal{A} & \textit{iff} & \textit{exists } Q.\ P \equiv (\boldsymbol{\nu}\overline{m})Q \textit{ and } Q \models_\varsigma \mathcal{A} \textit{ and } \overline{m}\# fn(\mathcal{A}) \\
P \models_\varsigma n : c(\mathcal{A})V & \textit{iff} & \textit{forall } R, Q.\ \textit{if } (P \|_* R) \textit{ and } R \models_\varsigma \mathcal{A} \textit{ and } P \mid R \Rightarrow_\varsigma Q \\
 & & \textit{then } \neg\mathrm{stuck}(Q) \textit{ and} \\
 & & \quad \textit{forall } Q', r.\ \textit{if } Q \overline{\overset{n.c(r)}{\longrightarrow}} Q' \textit{ then} \\
 & & \quad\quad \textit{exists } P', R', R_v.\ Q' \equiv P' \mid R' \mid R_v \textit{ and} \\
 & & \quad\quad\quad R_v \models_\varsigma r : V^\circ \textit{and } R \overset{\mathcal{A}}{\longmapsto}_\varsigma R'
\end{array}
$$

Fig. B.4. Sharing-indexed Satisfaction

$$
P \overset{\mathbf{0}}{\longmapsto}_\varsigma P \qquad
\frac{P \overset{\mathcal{U}}{\longmapsto}_\varsigma Q}{P \overset{\mathcal{U}\wedge\mathcal{V}}{\longmapsto}_\varsigma Q} \qquad
\frac{P \overset{\mathcal{V}}{\longmapsto}_\varsigma Q}{P \overset{\mathcal{U}\wedge\mathcal{V}}{\longmapsto}_\varsigma Q} \qquad
\frac{P \overset{\mathcal{U}\{^x/_n\}}{\longmapsto}_\varsigma Q}{P \overset{\forall x.\mathcal{U}}{\longmapsto}_\varsigma Q} \qquad
\frac{P \overset{n.\mathtt{1}_c(m)}{\longrightarrow} Q}{P \overset{n:\mathtt{1}_c(m)}{\longmapsto}_\varsigma Q}
$$

$$
\frac{P \equiv (\boldsymbol{\nu}\overline{m})R \quad R \overset{\mathcal{U}}{\longmapsto}_\varsigma Q}{P \overset{(\boldsymbol{\nu})\mathcal{U}}{\longmapsto}_\varsigma Q} \qquad
\frac{P \overset{\mathcal{U}}{\longmapsto}_\varsigma R \quad R \overset{\mathcal{V}}{\longmapsto}_\varsigma Q}{P \overset{\mathcal{U};\mathcal{V}}{\longmapsto}_\varsigma Q} \qquad
P \overset{\mathcal{U}^\circ}{\longmapsto}_\varsigma (P \setminus \mathcal{U}^\circ)
$$

$$
\frac{R \models_\varsigma \mathcal{A} \quad P \mid R \Rightarrow_\varsigma \overline{\overset{n.c(r)}{\longrightarrow}} Q \quad R \overset{\mathcal{A}}{\longmapsto}_\varsigma R'}{P \overset{n:c(\mathcal{A})V}{\longmapsto}_\varsigma Q \setminus R' \setminus r : V^\circ}
$$

$$
\frac{P \equiv P_1 \mid P_2 \quad P_1 \overset{\mathcal{U}}{\longmapsto}_\varsigma Q_1 \quad P_2 \overset{\mathcal{V}}{\longmapsto}_\varsigma Q_2 \quad Q_1 \mid Q_2 \equiv Q}{P \overset{\mathcal{U}\mid\mathcal{V}}{\longmapsto}_\varsigma Q}
$$

Fig. B.5. Usage

Theorem 4.4. If $P :: \mathbf{A} \shortmid \varsigma \triangleright \mathbf{B}$ then $P \models_\varsigma \mathbf{A} \triangleright \mathbf{B}$.

Proof. By induction on the typing derivation. We detail two interesting cases:

- (Case of (TSPar)) We have $P \mid Q :: \mathbf{A} \mid \mathbf{C} \mid !\mathbf{S} \shortmid \varsigma \triangleright \mathbf{B} \mid \mathbf{D}$ concluded from $P :: \mathbf{A} \mid !\mathbf{S} \shortmid \varsigma \triangleright \mathbf{B}$ and $Q :: \mathbf{C} \mid !\mathbf{S} \shortmid \varsigma \triangleright \mathbf{D}$. Pick $R \models_\varsigma \mathbf{A} \mid \mathbf{C} \mid !\mathbf{S}$. Then $R \equiv (\boldsymbol{\nu}\overline{m})(R_1 \mid R_2 \mid R_3)$ where $R_1 \models_\varsigma \mathbf{A}$ and $R_2 \models_\varsigma \mathbf{C}$ and $R_3 \models_\varsigma !\mathbf{S}$. Then $R_3 \models_\varsigma !\mathbf{S} \mid !\mathbf{S}$, so that $R_3 \equiv (\boldsymbol{\nu}\overline{n})(R_3^a \mid R_3^b)$ where $R_3^a \models_\varsigma !\mathbf{S}$ and $R_3^b \models_\varsigma !\mathbf{S}$. By i.h., we have that $P \mid R_1 \mid R_3^a \models_\varsigma \mathbf{B}$ and $Q \mid R_2 \mid R_3^b \models_\varsigma \mathbf{D}$. So, $R_1 \mid R_2 \mid P \mid Q \models_\varsigma \mathbf{B} \mid \mathbf{D}$. Then $R \mid P \mid Q \models_\varsigma \mathbf{B} \mid \mathbf{D}$.
- (Case of (TSObj)) Let $n[M \parallel \overline{s\langle n_i\rangle} \parallel t] :: \mathbf{A} \mid \Pi(n_i : V_i^\circ) \triangleright n : T$ be concluded from $[M;t] :: \mathbf{A} \shortmid \overline{s_i : V_i} : \ \triangleright \mathbf{B} \shortmid [T]$. By Lemma B.3, we have $n[M \parallel \overline{s\langle n_i\rangle} \parallel t] \models (\mathbf{A} \mid \Pi(n_i : V_i^\circ)) \triangleright (n : T);\mathbf{B}$. ■