Mobility Types for Mobile Ambients

Luca Cardelli
Microsoft Research

Giorgio Ghelli
Pisa University

Andrew D. Gordon
Microsoft Research

June 2, 1999

Technical Report
MSR-TR-99-32

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

A shortened version of this paper appears in the proceedings of the *International Conference on Automata, Languages, and Programming*, Prague, Czech Republic, 11–15 July 1999. The proceedings is published by Springer Verlag as a volume of the series *Lecture Notes in Computer Science*.

Mobility Types for Mobile Ambients

Luca Cardelli
Microsoft Research

Giorgio Ghelli
Pisa University

Andrew D. Gordon
Microsoft Research

June 2, 1999

Abstract

An ambient is a named cluster of processes and subambients, which moves as a group. The untyped ambient calculus is a process calculus in which ambients model a variety of concepts such as network nodes, packets, channels, and software agents. In these models, some ambients are intended to be mobile, some immobile; and some are intended to be ephemeral, some persistent. We describe type systems able to formalize these intentions: they can guarantee that an ambient will remain immobile, and that an ambient will not be dissolved by its environment. These guarantees could help establish security properties of models, for instance. A novel feature of our type systems is their distinction between mobile and immobile processes.

Contents

1 Motivation

The ambient calculus [CG98] is a process calculus that focuses primarily on process mobility rather than process communication. An ambient is a named location that may contain processes and subambients, and that can move as a unit inside or outside other ambients. Processes within an ambient may cause their enclosing ambient to move, and may communicate by anonymous asynchronous messages dropped into the local ether. Moreover, processes may open subambients, meaning that they can dissolve an ambient boundary and cause the contents of that ambient to spill into the parent ambient. The ability to move and open ambients is regulated by capabilities that processes must possess by prior knowledge or acquire by communication.

In earlier work [CG99] we studied type systems for the ambient calculus that control the exchange of values during communication. Those type systems are designed to match the communication primitives of the ambient calculus, but are able to express familiar typings for processes and functions. They are therefore successful in showing that the typed ambient calculus is as expressive as typed process and function calculi. Still, those type systems say nothing about process mobility: they guarantee that communication is well-typed wherever it may happen, but do not constrain the movement of ambients.

In this paper we study type systems that control the movement of ambients through other ambients. Our general aim is to discover type systems that can be useful for constraining the mobility behavior of agents and other entities that migrate over networks. Guarantees provided by a type system for mobility could then be used for security purposes, in the sense exemplified by Java bytecode verification [LY97]. The idea of using a type system to constrain dynamic behavior is certainly not new, but this paper makes two main contributions. First, we exhibit type systems that constrain whether or not an ambient is mobile, and whether or not an ambient may be opened. Although previous authors [AP94, RH98, Sew98] use syntactic constraints to determine whether or not a process or location can move, the use of typing to draw this distinction appears to be new. Second, we propose a new mobility primitive as a solution to a problem of unwanted propagation of mobility effects from mobile ambients to those intended to be immobile.

Section 2 describes our system of mobility and locking annotations, and illustrates it using the example of encoding communication channels. The type system of [CG99] is included in the one presented here, and we survey it

in passing. In Section 3 we discuss the mobility primitive mentioned above; it is derivable in the untyped calculus, but not in the system of Section 2, and hence needs to be typed specially. Section 4 examines typed encodings of a distributed language that supports thread migration between hosts. Section 5 concludes and surveys related work.

2 Mobility and Locking Annotations

This section explains our basic type system for mobility, which directly extends our previous untyped and typed calculi.

2.1 An Example of Ambient Behavior (Review)

Although we assume in this paper some familiarity with the untyped ambient calculus [CG98], we begin by reviewing its main features by example.

The process

$$a[p[\mathit{out}\ a.\mathit{in}\ b.\langle M\rangle]] \mid b[\mathit{open}\ p.(x).Q]$$

models the movement of a packet p, which contains a message M, from location a to location b. The process $p[\mathit{out}\ a.\mathit{in}\ b.\langle M\rangle]$ is an ambient named p that contains a single process $\mathit{out}\ a.\mathit{in}\ b.\langle M\rangle$. It is the only subambient of the ambient named a, which itself has a sibling ambient $b[\mathit{open}\ p.(x).Q]$. The terms $\mathit{out}\ a$, $\mathit{in}\ b$, and $\mathit{open}\ p$ are capabilities, which processes exercise to cause ambients to move or to be opened.

In this example, the process $\mathit{out}\ a.\mathit{in}\ b.\langle M\rangle$ exercises the capability $\mathit{out}\ a$, which causes its enclosing ambient, the one named p, to exit its own parent, the one named a, so that $p[\mathit{in}\ b.\langle M\rangle]$ now runs in parallel with the ambients a and b. Next, the process $\mathit{in}\ b.\langle M\rangle$ causes the ambient p to enter b, so that $p[\langle M\rangle]$ becomes a subambient of b. Up to this point, the process $\mathit{open} p.(x).Q$ was blocked, but now $\mathit{open}\ p$ can dissolve the boundary p. Finally, the input $(x).Q$ consumes the output $\langle M\rangle$, to leave the residue $a[] \mid b[Q\{x{\leftarrow}M\}]$, where $Q\{x{\leftarrow}M\}$ is the outcome of replacing each occurrence of x in Q with the expression M.

Two additional primitives of our calculus are replication and restriction. Just as in the π-calculus [Mil91], a replication $!P$ behaves the same as an infinite array of replicas of P running in parallel, and a restriction $(\nu n)P$ means: pick a completely fresh name, call it n, then run P.

2.2 Syntax and Operational Semantics

We recall the syntax of the typed ambient calculus from [CG99]. This is the same syntax as the original untyped ambient calculus [CG98], except that type annotations are added to the ν and input constructs, and that input and output are polyadic. We explain the types that appear in the syntax in the next section.

Expressions and Processes:

$M ::=$	expression
n	name
$in\ M$	enter into M
$out\ M$	exit out of M
$open\ M$	open M
ϵ	null
$M.M'$	path
$P, Q, R ::=$	process
$(\nu n{:}W)P$	restriction
$\mathbf{0}$	inactivity
$P \mid Q$	composition
$!P$	replication
$M[P]$	ambient
$M.P$	action
$(x_1{:}W_1, \ldots, x_k{:}W_k).P$	input action
$\langle M_1, \ldots, M_k \rangle$	async output action

We identify processes up to the consistent renaming of bound names. We write $P\{n \leftarrow M\}$ for the substitution of the term M for each free occurrence of the name n in process P. We often use the notations M and $n[]$ as shorthands for the processes $M.\mathbf{0}$ and $n[\mathbf{0}]$, respectively.

A structural equivalence relation $P \equiv Q$ identifies certain processes P and Q whose behavior ought always to be equivalent:

Structural congruence:

$P \equiv P$	(Struct Refl)
$Q \equiv P \Rightarrow P \equiv Q$	(Struct Symm)

$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow (\nu n{:}W)P \equiv (\nu n{:}W)Q$	(Struct Res)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(Struct Amb)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(Struct Action)
$P \equiv Q \Rightarrow$ $(x_1{:}W_1, \ldots, x_k{:}W_k).P \equiv (x_1{:}W_1, \ldots, x_k{:}W_k).Q$	(Struct Input)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!P \equiv P \mid !P$	(Struct Repl Par)
$n_1 \neq n_2 \Rightarrow$ $(\nu n_1{:}W_1)(\nu n_2{:}W_2)P \equiv (\nu n_2{:}W_2)(\nu n_1{:}W_1)P$	(Struct Res Res)
$n \notin fn(P) \Rightarrow$ $(\nu n{:}W)(P \mid Q) \equiv P \mid (\nu n{:}W)Q$	(Struct Res Par)
$n \neq m \Rightarrow (\nu n{:}W)m[P] \equiv m[(\nu n{:}W)P]$	(Struct Res Amb)
$P \mid \mathbf{0} \equiv P$	(Struct Zero Par)
$(\nu n{:}Amb^Y[^ZT])\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Res)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Repl)
$\epsilon.P \equiv P$	(Struct ϵ)
$(M.M').P \equiv M.M'.P$	(Struct .)

We specify process behavior via a reduction relation, $P \rightarrow Q$. The first four rules describe the effects of, respectively, *in*, *out*, *open*, and communication.

Reduction:

$n[in\ m.P \mid Q] \mid m[R]$	$\rightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.P \mid Q] \mid R]$	$\rightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.P \mid n[Q]$	$\rightarrow P \mid Q$	(Red Open)
$\langle M_1, \ldots, M_k \rangle \mid (x_1{:}W_1, \ldots, x_k{:}W_k).P$	$\rightarrow P\{x_1 \leftarrow M_1, \ldots, x_k \leftarrow M_k\}$	(Red I/O)
$P \rightarrow Q$	$\Rightarrow P \mid R \rightarrow Q \mid R$	(Red Par)
$P \rightarrow Q$	$\Rightarrow (\nu n{:}W)P \rightarrow (\nu n{:}W)Q$	(Red Res)
$P \rightarrow Q$	$\Rightarrow n[P] \rightarrow n[Q]$	(Red Amb)
$P' \equiv P, P \rightarrow Q, Q \equiv Q'$	$\Rightarrow P' \rightarrow Q'$	(Red $\equiv$)

2.3 The Type System

The basic type constructions from [CG99] are the ambient types $Amb[T]$ and the capability types $Cap[T]$. A type of the form $Amb[T]$ describes names that name ambients that allow the exchange of T information within. A type of the form $Cap[T]$ is used to track the opening of ambients: it describes capabilities that may cause the unleashing of T exchanges by means of opening subambients into the current one. An exchange is the interaction of an input and an output operation within the local ether of an ambient. The exchange types, T, can be either *Shh* (no exchange allowed) or a tuple type where each component describes either a name or a capability.

In this paper, we enrich these types with two attributes indicating whether an ambient can move at all, and whether it can be opened. These attributes are intended as two of the simplest properties one can imagine that are connected with mobility. (In another paper [CGG99] we investigate more expressive and potentially more useful generalizations of these attributes.)

We first describe the locking attributes, Y. An ambient can be declared to be either locked ($\bullet$) or unlocked ($\circ$). Locked ambients can never be opened, while unlocked ambients can be opened via an appropriate capability. The locking attributes are attached to the $Amb[T]$ types, which now acquire the form $Amb^Y[T]$. This means that any ambient whose name has type $Amb^Y[T]$ may (or may not) be opened, and if opened may unleash T exchanges.

We next describe the mobility attributes, Z. In general, a process can produce a number of effects that may be tracked by a type system. Previously we tracked only communication effects, T. We now plan to track both mobility and communication effects by pairs of the form ${}^ZT$, where Z is a flag indicating that a process executes movement operations ($\frown$) or does not ($\overset{\vee}{-}$), and T is as before. A process with effects ${}^ZT$ should be allowed to run only within a compatible ambient, whose type will therefore have the form $Amb[{}^ZT]$. A capability, when used, may now cause communication effects or mobility effects, and its type will have the form $Cap[{}^ZT]$. An *in* or an *out* capability directly causes a mobility effect, whereas an *open* capability may indirectly cause both communication and mobility effects.

The following table describes the syntax of our types. An ambient type $Amb^Y[{}^ZT]$ describes the name of an ambient whose locking and mobility attributes are Y and Z, respectively, and which allows T exchanges.

Types:

$Y ::=$	locking annotation
$\bullet$	locked
$\circ$	unlocked
$Z ::=$	mobility annotation
$\curvearrowright$	mobile
$\underline{\vee}$	immobile
$W ::=$	message type
$Amb^{Y}[^{Z}T]$	ambient name
$Cap[^{Z}T]$	capability
$T ::=$	exchange type
Shh	no exchange
$W_1 \times \cdots \times W_k$	tuple exchange

The type rules are formally described in the next tables. There are three typing judgments: the first constructs well-formed environments, the second tracks the types of messages, and the third tracks the effects of processes. The rules for *in* and *out* introduce mobility effects, and the rule for *open* requires unlocked ambients. The handling of communication effects, T, is exactly as in [CG99].

Judgments:

$E \vdash \diamond$	good environment
$E \vdash M : W$	good expression of message type W
$E \vdash P : {}^{Z}T$	good process with mobility Z exchanging T

Good environment:

(Env $\varnothing$)
$$\frac{}{\varnothing \vdash \diamond}$$

(Env n)
$$\frac{E \vdash \diamond \quad n \notin dom(E)}{E, n{:}W \vdash \diamond}$$

Good expression of type W:

(Exp n)
$$\frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W}$$

(Exp ϵ)
$$\frac{E \vdash \diamond}{E \vdash \epsilon : Cap[^{Z}T]}$$

(Exp .)
$$\frac{E \vdash M : Cap[^{Z}T] \quad E \vdash M' : Cap[^{Z}T]}{E \vdash M.M' : Cap[^{Z}T]}$$

(Exp In)
$$\frac{E \vdash n : Amb^{Y}[^{Z}T]}{E \vdash in\, n : Cap[^{\frown}T']}$$

(Exp Out)
$$\frac{E \vdash n : Amb^{Y}[^{Z}T]}{E \vdash out\, n : Cap[^{\frown}T']}$$

(Exp Open)
$$\frac{E \vdash n : Amb^{\circ}[^{Z}T]}{E \vdash open\, n : Cap[^{Z}T]}$$

Good process with mobility Z exchanging T:

(Proc Action)
$$\frac{E \vdash M : Cap[^{Z}T] \quad E \vdash P : {}^{Z}T}{E \vdash M.P : {}^{Z}T}$$

(Proc Amb)
$$\frac{E \vdash M : Amb^{Y}[^{Z}T] \quad E \vdash P : {}^{Z}T}{E \vdash M[P] : {}^{Z'}T'}$$

(Proc Res)
$$\frac{E, n{:}Amb^{Y}[^{Z}T] \vdash P : {}^{Z'}T'}{E \vdash (\nu n{:}Amb^{Y}[^{Z}T])P : {}^{Z'}T'}$$

(Proc Par)
$$\frac{E \vdash P : {}^{Z}T \quad E \vdash Q : {}^{Z}T}{E \vdash P \mid Q : {}^{Z}T}$$

(Proc Repl)
$$\frac{E \vdash P : {}^{Z}T}{E \vdash !P : {}^{Z}T}$$

(Proc Input)
$$\frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : {}^{Z}W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : {}^{Z}W_1 \times \cdots \times W_k}$$

(Proc Output)
$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k}{E \vdash \langle M_1, \ldots, M_k \rangle : {}^{Z}W_1 \times \cdots \times W_k}$$

(Proc Zero)
$$\frac{E \vdash \diamond}{E \vdash \mathbf{0} : {}^{Z}T}$$

For example, consider the untyped process discussed in Section 2.1:

$$a[p[out\, a.in\, b.\langle M \rangle]] \mid b[open\, p.(x).Q]$$

Suppose that the message M has type W. We can type the process under the assumption that a is a locked, immobile ambient ($Amb^{\bullet}[^{\veebar}Shh]$), that p is an unlocked, mobile ambient ($Amb^{\circ}[^{\frown}W]$), and that b is a locked, mobile ambient ($Amb^{\bullet}[^{\frown}W]$). More formally, under the assumptions $E \vdash a : Amb^{\bullet}[^{\veebar}Shh]$, $E \vdash p : Amb^{\circ}[^{\frown}W]$, $E \vdash b : Amb^{\bullet}[^{\frown}W]$, $E \vdash M : W$, and $E, x{:}W \vdash P : {}^{\frown}W$ we can derive that $E \vdash a[p[out\, a.in\, b.\langle M \rangle]] \mid b[open\, p.(x{:}W).P] : {}^{\veebar}Shh$. Note that the ambient b cannot be annotated as immobile, because within it, the mobile ambient p is opened.

Moreover, here are four examples of processes that are not typable. Processes (1) and (2) violate a mobility assumption. Process (3) violates a locking assumption, while process (4) violates a communication assumption.

(1) $(x{:}Amb^{Y}[^{\vee}_{-}T]).x[out\ m]$

(2) $(\nu n{:}Amb^{Y}[^{\vee}_{-}T])(n[open\ m] \mid m[in\ n.out\ p])$

(3) $(\nu n{:}Amb^{\bullet}[^{Z}T])(n[] \mid \langle n \rangle \mid (x{:}Amb^{\bullet}[^{Z}T]).open\ x)$

(4) $(\nu n{:}Amb^{\bullet}[^{Z}T])(n[] \mid \langle n \rangle \mid (x{:}Amb^{\circ}[^{Z}T]).open\ x)$

As customary, we can prove a subject reduction theorem asserting the soundness of the typing rules. It can be interpreted as stating that every communication is well-typed, that no locked ambient will ever be opened, and that no *in* or *out* will ever act on an immobile ambient. As in earlier work [CG99], the proof is by induction on derivations.

Theorem 1 *If* $E \vdash P : {}^{Z}T$ *and* $P \to Q$ *then* $E \vdash Q : {}^{Z}T$.

Proof See Appendix A. □

Remark 1 *The type system of [CG99] can be embedded in the current type system by taking* $Amb[T] = Amb^{\circ}[^{\frown}T]$ *and* $Cap[T] = Cap[^{\frown}T]$.

2.4 Encoding Channels

Communication in the basic ambient calculus happens in the local ether of an ambient. Messages are simply dropped into the ether, without specifying a recipient other than any process that does or will exist in the current ambient. Even within the ambient calculus, though, one often feels the need of additional communication operations, whether primitive or derived.

The familiar mechanism of communication over named channels, used by most process calculi, can be expressed fairly easily in the untyped ambient calculus. We should think, though, of a channel as a new entity that may reside within an ambient. In particular, communications executed on the same channel name but in separate ambients will not interact, at least until those ambients are somehow merged.

The basic idea for representing channels is as follows; see [CG98, CG99] for details. If c is the name of a channel we want to represent, then we use

a name c^b to name an ambient that acts as a communication buffer for c. We also use a name c^p to name ambients that act as communication packets directed at c. The buffer ambient opens all the incoming packet ambients and lets their contents interact. So, an output on channel c is represented as a c^p packet that enters c^b (where it is opened up) and that contains an output operation. Similarly, an input on channel c is represented as a c^p packet that enters c^b (where it is opened up) and that contains an input operation; after the input is performed, the rest of the process exits the buffer appropriately to continue execution. The creation of a channel name c is represented as the creation of the two names c^b and c^p. Similarly, the communication of a channel name c is represented as the communication of the two names c^b and c^p.

This encoding of channels can be typed within the type system of [CG99]. Let $Ch[T]$ denote the type of a channel c exchanging messages of type T. This type can be represented as $Amb[T] \times Amb[T]$, which is the type of the pair of names c^b, c^p. Packets named c^p have exchange type T by virtue of performing corresponding inputs and outputs. Buffers named c^p have exchange type T by virtue of opening c^p packets, and unleashing their exchanges.

The natural question now is whether we can type this encoding of channels in the type system given earlier. This can be done trivially by Remark 1, by making all the ambients movable and openable. But this solution is not very satisfactory. In particular, now that we have a type system for mobility, we would like to declare the communication buffers to be both immobile and locked, so that channel communication cannot be disrupted by accidental or malicious activities. Note, for example, that a malicious packet could contain instructions that would cause the buffer to move when the packet is opened. Such a packet should be ruled out as untypable if we made the buffer immobile.

The difficulty with protecting buffers from malicious packets does not arise if we use a systematic translation of a high-level channel abstraction into the lower-level ambient calculus. However, in a situation where code is untrusted (for example, mobile code received from the network), we cannot assume that the ambient-level code interacting with the channel buffers is the image of a high-level abstraction. Thus, we would like to typecheck the untrusted code to make sure that it satisfies the mobility constraints of the trusted environment.

We now encounter a fundamental difficulty that will haunt us for the rest of this section and all of the next. In the type system given earlier, we cannot

declare buffers to be immobile, because buffers open packets that are mobile; therefore, buffers are themselves potentially mobile. Packets must, of course, be mobile because they must enter buffers.

We have explored several possible solutions to this problem. In the rest of this section we present a different (more complex) encoding of channels that satisfies several of our wishes. In the next section we add a typed primitive that allows us to use an encoding similar to the original one; this new primitive has other applications as well. In addition, one could investigate more complex type systems that attempt to capture the fact that a well-behaved packet moves once and then becomes immobile, or some suitable generalization of this notion.

The idea for the encoding shown below comes from [CG99], where an alternative encoding of channels is presented. In that encoding there are no buffers; the packets, though, are self-coalescing, so that each packet can act as an exchange buffer for another packet. Here we combine the idea of self-coalescing packets with an immobile buffer that contains them. Since nothing is opened directly within the buffer, the difficulty with constraining the mobility of buffers, described above, disappears. A trace of the difficulty, though, remains in that a process performing an input must be given a mobile type, even when it performs only channel communications.

We formalize our encoding of channels by considering a calculus obtained from the system given earlier by adding operations for creating typed channels ($(\nu c{:}Ch[C_1, \ldots, C_k])P$ where the C_i's are channel types) and for typed inputs and outputs over them ($c\langle n_1, \ldots, n_k\rangle$ and $c(x_1{:}C_1, \ldots, x_k{:}C_k).P$, where c is the channel name, the n_i's are other channel names that are communicated over it, and the C_i's are channel types).

We assemble the extended calculus by adding a new syntactic category of channel types, C, and by extending the syntactic category of processes:

An ambient calculus with channels:

$C ::=$	channel type
$\quad Ch[C_1, \ldots, C_k]$	$\quad$ channel carrying tuples of channels
$P, Q, R ::=$	process
$\quad \ldots$	$\quad$ as in Section 2.2
$\quad (\nu c{:}C)P$	$\quad$ new channel
$\quad c\langle n_1, \ldots, n_k\rangle$	$\quad$ output on a channel
$\quad (x_1{:}C_1, \ldots, x_k{:}C_k).P$	$\quad$ input off a channel

The syntax of locking and mobility annotations, and message and exchange types is unchanged.

We write the three judgments of the type system of the extended calculus as $E \vdash_{ch} \diamond$, $E \vdash_{ch} M : W$, and $E \vdash_{ch} P : {}^{Z}T$. We write $\vdash_{ch}$ instead of $\vdash$ to distinguish these judgments from those of the original calculus. The new judgments are defined by the same rules as before together with the following:

Channel I/O:

$$\frac{E \vdash_{ch} \diamond \quad c \notin dom(E)}{E, c{:}C \vdash_{ch} \diamond} \text{(Env } c\text{)} \qquad \frac{E, n{:}C \vdash_{ch} P : {}^{Z}T}{E \vdash_{ch} (\nu n{:}C)P : {}^{Z}T} \text{(Ch Res)}$$

$$\frac{E \vdash_{ch} c : Ch[C_1, \ldots, C_k] \quad E, x_1{:}C_1, \ldots, x_k{:}C_k \vdash_{ch} P : {}^{Z}T \quad Z = \curvearrowright}{E \vdash_{ch} c(x_1{:}C_1, \ldots, x_k{:}C_k).P : {}^{Z}T} \text{(Ch Input)}$$

$$\frac{E \vdash_{ch} c : W \quad E \vdash_{ch} M_1 : C_1 \quad \cdots \quad E \vdash_{ch} M_k : C_k}{E \vdash_{ch} c\langle M_1, \ldots, M_k \rangle : {}^{Z}T} \text{(Ch Output)}$$

Since the syntaxes of message and channel types are independent, ether I/O can only communicate ambient names and capabilities, and channel I/O can only communicate channel names. There is a more liberal variant of the extended calculus in which the syntax of message and channel types are combined, and both kinds of I/O can communicate all three kinds of data. We prefer the version above to the more liberal variant, because the translation of the liberal variant into the ambient calculus without channels is more complicated than the translation of the version above, but no more informative.

The following tables describe a translation of the extended calculus into the one without channels. In the translation, a packet sent or received on a channel c is encoded by an ambient named c^p whose type is mobile and unlocked, but the channel itself is encoded by an ambient named c^b whose type is immobile and locked. Therefore, the type system guarantees that the channel cannot be tampered with by rogue processes.

Translation of the extended calculus:

$[\![E]\!]$	environment obtained from E
$[\![C]\!]^b$	type for buffers representing channels of type C
$[\![C]\!]^p$	type for packets on channels of type C
$[\![P]\!]$	process obtained from P

Translation of environments:

$$[\![\varnothing]\!] \triangleq \varnothing$$
$$[\![E, n{:}W]\!] \triangleq [\![E]\!], n{:}W$$
$$[\![E, c{:}C]\!] \triangleq [\![E]\!], c^b{:}[\![C]\!]^b, c^p{:}[\![C]\!]^p$$

Translation of channel types:

$$[\![\, Ch[C_1, \ldots, C_k]]\!]^b \triangleq Amb^\bullet[^\vee Shh]$$
$$[\![\, Ch[C_1, \ldots, C_k]]\!]^p \triangleq Amb^\circ[\frown [\![C_1]\!]^b \times [\![C_1]\!]^p \times \cdots \times [\![C_k]\!]^b \times [\![C_k]\!]^p]$$

Translation of processes:

$$[\![(\nu n{:}W)P]\!] \triangleq (\nu n{:}W)[\![P]\!]$$
$$[\![\mathbf{0}]\!] \triangleq \mathbf{0}$$
$$[\![P \mid Q]\!] \triangleq [\![P]\!] \mid [\![Q]\!]$$
$$[\![!P]\!] \triangleq ![\![P]\!]$$
$$[\![M[P]]\!] \triangleq M[[\![P]\!]]$$
$$[\![M.P]\!] \triangleq M.[\![P]\!]$$
$$[\![(x_1{:}W_1, \ldots, x_k{:}W_k).P]\!] \triangleq (x_1{:}W_1, \ldots, x_k{:}W_k).[\![P]\!]$$
$$[\![\langle M_1, \ldots, M_k\rangle]\!] \triangleq \langle M_1, \ldots, M_k\rangle$$
$$[\![(\nu c{:}C)P]\!] \triangleq (\nu c^b{:}[\![C]\!]^b)(\nu c^p{:}[\![C]\!]^p)(c^b[\,] \mid [\![P]\!])$$
$$[\![c\langle n_1, \ldots, n_k\rangle]\!] \triangleq c^p[in\ c^b.(!open\ c^p \mid in\ c^p \mid \langle n_1^b, n_1^p, \ldots, n_k^b, n_k^p\rangle)]$$
$$[\![c(x_1{:}C_1, \ldots, x_k{:}C_k).P_{\frown T}]\!] \triangleq$$
$$\quad (\nu s{:}Amb^\circ[\frown T])$$
$$\quad\quad (open\ s \mid c^p[in\ c^b.(!open\ c^p \mid in\ c^p \mid$$
$$\quad\quad\quad\quad (x_1^b{:}[\![C_1]\!]^b, x_1^p{:}[\![C_1]\!]^p, \ldots, x_k^b{:}[\![C_k]\!]^b, x_k^p{:}[\![C_k]\!]^p).$$
$$\quad\quad\quad\quad\quad s[!out\ c^p \mid out\ c^b.[\![P]\!]])])$$
$$\quad \text{for } s \notin \{c^b, c^p, \} \cup fn([\![P]\!])$$

(The translation $[\![c(x_1{:}C_1, \ldots, x_k{:}C_k).P_{\frown T}]\!]$ depends on the type of the process P, which we indicate by the subscript $\frown T$.)

Proposition 1

(1) *If* $E \vdash_{ch} \diamond$ *then* $[\![E]\!] \vdash \diamond$.

(2) *If* $E \vdash_{ch} M : W$ *then* $[\![E]\!] \vdash M : W$.

(3) *If* $E \vdash_{ch} c : C$ *then* $[\![E]\!] \vdash c^b : [\![C]\!]^b$ *and* $[\![E]\!] \vdash c^p : [\![C]\!]^p$.

(4) *If* $E \vdash_{ch} P :^Z T$ *then* $[\![E]\!] \vdash [\![P]\!] :^Z T$.

Proof By inductions on the derivations. □

In summary, this translation demonstrates a typing of channels in which channels are immobile ambients. However, a feature of this typing is that in an input $c(x_1{:}W_1, \ldots, x_k{:}W_k).P$, the process P is obliged to be mobile. The next section provides a type system that removes this obligation.

3 Objective moves

3.1 Subjective versus Objective Moves

The movement operations of the standard ambient calculus are called "subjective" because they have the flavor of "I (ambient) wish to move there". Other movement operations are called "objective" when they have the flavor of "you (ambient) should move there". Objective moves can be adequately emulated with subjective moves [CG98]: the latter were chosen as primitive on the grounds of expressive power and simplicity.

Certain objective moves, however, can acquire additional interpretations with regard to the typing of mobility. In this section we introduce objective moves, and we distinguish between subjective-mobility annotations (the ones of Section 2) and objective-mobility annotations. It is perhaps not too surprising that the introduction of typing constructs requires the introduction of new primitives. For example, in both the π-calculus and the ambient calculus, the introduction of simple types requires a switch from monadic to polyadic I/O.

We consider an objective move operation that moves to a different location an ambient that has not yet started. It has the form $go\, N.M[P]$ and has

the effect of starting the ambient $M[P]$ in the location reached by following, if possible, the path N. Note that P does not become active until after the movement is completed.

Unlike *in* and *out*, this *go* operation does not move the ambient enclosing the operation. Possible interpretations of this operation are to install a piece of code at a given location and then run it, or to move the continuation of a process to a given location.

When assigning a mobility type to the *go* operation, we can now make a subtle distinction. The ambient $M[P]$ is moved, objectively, from one place to another. But after it gets there, maybe the process P never executes subjective moves, and therefore M can be declared subjectively immobile. Moreover, the *go* operation itself does not cause its surrounding ambient to move, so it may also be possible to declare the surrounding ambient subjectively immobile.

Therefore, we can move an ambient from one place to another without noticing any subjective-mobility effects. Still, something got moved, and we would like to be able to track this fact in the type system. For this purpose, we introduce objective-mobility annotations, attached to ambients that may be objectively moved. In particular, an ambient may be objectively mobile, but subjectively immobile.

In conclusion, we achieve the task, impossible in the type system of Section 2, of moving an immobile ambient, once. (More precisely, the possible encodings of the *go* operation in terms of subjective moves are not typable in the type system of Section 2 if we set M to be immobile.) The additional expressive power can be used to give a better typing to communication channels, by causing a communication packet to move into a buffer without requiring the packet to be itself subjectively mobile, and therefore without having to require the buffer that opens the packet to be subjectively mobile.

3.2 Syntax and Operational Semantics

To formalize these ideas, we make the following changes to the system of Section 2. Using objective moves we can type an encoding of channels which eliminates the immobility obligation noted at the end of the previous section. Moreover, in Section 4, objective moves are essential for encoding an example language, in which mobile threads migrate between immobile hosts.

Processes:

$P, Q, R ::=$	process
$\ldots$	as in Section 2.2
$go\ N.M[P]$	objective move

Structural congruence:

$P \equiv Q \Rightarrow go\ N.M[P] \equiv go\ N.M[Q]$	(Struct Go)
$(\nu n{:}Amb^{Y\,Z'}[^{Z}T])\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Res)
$go\ \epsilon.M[P] \equiv M[P]$	(Struct Go ϵ)

Reduction:

$go\ (in\ m.N).n[P] \mid m[Q] \rightarrow m[go\ N.n[P] \mid Q]$	(Red Go In)
$m[go\ (out\ m.N).n[P] \mid Q] \rightarrow go\ N.n[P] \mid m[Q]$	(Red Go Out)

Objective moves can be mimicked in the untyped ambient calculus by taking $goN.M[P]$ to be a shorthand for the untyped process $(\nu k)k[N.M[outk.P]]$ for k not free in $N.M[P]$. This encoding is typable in the type system of the previous section, but only under the restriction that the process P is subjectively mobile. We propose primitive typing rules for objective moves so that P may be subjectively immobile.

3.3 The Type System

The types of the system extended with objective moves are the same as the types in Section 2, except that the types of ambient names are $Amb^{Y\,Z'}[^{Z}T]$, where Y is a locking annotation, T is an exchange type, and Z' and Z are an objective-mobility annotation and a subjective-mobility annotation, respectively.

Types:

$Y ::=$	ambient locking
$\bullet$	locked
$\circ$	unlocked
$Z ::=$	mobility effects

$\curvearrowright$	mobile
$\underline{\vee}$	immobile
$W ::=$	message types
$Amb^{Y\,Z'}[^{Z}T]$	ambient name
$Cap[^{Z}T]$	capability
$T ::=$	exchange types
Shh	no exchange
$W_1 \times \cdots \times W_k$	tuple exchange

The typing rules of the extended system are similar to the rules from Section 2. The only interesting use of the objective-mobility annotations is in the one new rule, (Proc Go), which types objective moves: (Proc Go) constrains the objective-mobility annotation, Z', of the ambient to be the same as the mobility effect of the capability, N. So, if N is $in\,n$ or $out\,n$, say, Z' must be $\curvearrowright$, the mobility effect of N. On the other hand, (Proc Go) does not constrain the subjective-mobility annotation, Z, of the ambient. So the process P within the ambient may be either mobile or immobile.

Judgments:

$E \vdash \diamond$	good environment
$E \vdash M : W$	good expression of message type W
$E \vdash P : {}^{Z}T$	process with mobility Z exchanging T

Good expressions:

(Exp n)
$$\frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W}$$

(Exp ϵ)
$$\frac{E \vdash \diamond}{E \vdash \epsilon : Cap[^{Z}T]}$$

(Exp .)
$$\frac{E \vdash M : Cap[^{Z}T] \quad E \vdash M' : Cap[^{Z}T]}{E \vdash M.M' : Cap[^{Z}T]}$$

(Exp In)
$$\frac{E \vdash n : Amb^{Y\,Z'}[^{Z}T]}{E \vdash in\,n : Cap[^{\curvearrowright}T']}$$

(Exp Out)
$$\frac{E \vdash n : Amb^{Y\,Z'}[^{Z}T]}{E \vdash out\,n : Cap[^{\curvearrowright}T']}$$

(Exp Open)
$$\frac{E \vdash n : Amb^{\circ\,Z'}[^{Z}T]}{E \vdash open\,n : Cap[^{Z}T]}$$

Good processes:

(Proc Action)
$$\frac{E \vdash M : Cap[^{Z}T] \quad E \vdash P : {}^{Z}T}{E \vdash M.P : {}^{Z}T}$$

(Proc Amb)
$$\frac{E \vdash M : Amb^{Y\,Z''}[^{Z}T] \quad E \vdash P : {}^{Z}T}{E \vdash M[P] : {}^{Z'}T'}$$

(Proc Res)
$$\frac{E, n{:}Amb^{Y\,Z''}[^{Z}T] \vdash P : {}^{Z'}T'}{E \vdash (\nu n{:}Amb^{Y\,Z''}[^{Z}T])P : {}^{Z'}T'}$$

(Proc Zero)
$$\frac{E \vdash \diamond}{E \vdash \mathbf{0} : {}^{Z}T}$$

(Proc Par)
$$\frac{E \vdash P : {}^{Z}T \quad E \vdash Q : {}^{Z}T}{E \vdash P \mid Q : {}^{Z}T}$$

(Proc Repl)
$$\frac{E \vdash P : {}^{Z}T}{E \vdash !P : {}^{Z}T}$$

(Proc Input)
$$\frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : {}^{Z}W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : {}^{Z}W_1 \times \cdots \times W_k}$$

(Proc Output)
$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k}{E \vdash \langle M_1, \ldots, M_k \rangle : {}^{Z}W_1 \times \cdots \times W_k}$$

(Proc Go)
$$\frac{E \vdash N : Cap[^{Z'}S] \quad E \vdash M : Amb^{Y\,Z'}[^{Z}T] \quad E \vdash P : {}^{Z}T}{E \vdash go\ N.M[P] : {}^{Z''}T''}$$

Theorem 2 *If $E \vdash P : {}^{Z}T$ and $P \to Q$ then $E \vdash Q : {}^{Z}T$.*

Proof See Appendix B. □

Objective moves allow us to type the example of sending a packet from ambient a to ambient b more flexibly than in the previous section. We rewrite the example to use object moves:

$$a[go\ (out\ a.in\ b).p[\langle M \rangle]] \mid b[open\ p.(x{:}W).P]$$

Then, if we assume $E \vdash M : W$, $E \vdash a : Amb^{\bullet\,{}^{\vee}_{-}}[{}^{\vee}_{-}Shh]$, $E \vdash p : Amb^{\circ\frown}[{}^{\vee}_{-}W]$, $E \vdash b : Amb^{\bullet\,{}^{\vee}_{-}}[{}^{\vee}_{-}W]$, $E, x{:}W \vdash P : {}^{\vee}_{-}W$ we get the typing:

$$E \vdash a[go\ (out\ a.in\ b).p[\langle M \rangle]] \mid b[open\ p.(x{:}W).P] : {}^{\vee}_{-}Shh$$

Unlike the situation in Section 2.3, ambients a and b can be annotated as both objectively and subjectively immobile. This is desirable if a and b are intended to represent immobile hosts on a network.

3.4 Encoding Channels, Again

Now, recall the extension of our previous system with channel-based communication. We can extend our new system with channels in a similar way. The new rules for channels are exactly as in Section 2, except that this time we can drop the side-condition $Z = \curvearrowright$ in the typing rule for channel input, (Ch Input). We can adapt our translation from Section 2.4 to exploit objective moves; here are the clauses that change:

Expressing channels with ambients, using objective moves:

$$[\![Ch[C_1, \ldots, C_k]]\!]^b \triangleq Amb^\circ[\curvearrowright [\![C_1]\!] \times \ldots \times [\![C_k]\!]]$$
$$[\![Ch[C_1, \ldots, C_k]]\!]^p \triangleq Amb^\bullet[\curvearrowright [\![C_1]\!] \times \ldots \times [\![C_k]\!]]$$
$$[\![(\nu c{:}C)P]\!] \triangleq (\nu c^b{:}[\![C]\!]^b)(\nu c^p{:}[\![C]\!]^p)(c^b[!open\ c^p] \mid [\![P]\!])$$
$$[\![c\langle n_1, \ldots, n_k\rangle]\!] \triangleq go\ in\ c^b.c^p[\langle n_1^b, n_1^p, \ldots, n_k^b, n_k^p\rangle]$$
$$[\![c(x_1{:}C_1, \ldots, x_k{:}C_k).P_{z_T}]\!] \triangleq$$
$$(\nu s{:}Amb^\circ\curvearrowright[^Z T])$$
$$(open\ s \mid$$
$$go\ in\ c^b.c^p[(x_1^b{:}[\![C_1]\!]^b, x_1^p{:}[\![C_1]\!]^p, \ldots, x_k^b{:}[\![C_k]\!]^b, x_k^p{:}[\![C_k]\!]^p).go\ out\ c^b.s[[\![P]\!]]])$$
$$\text{for } s \notin \{c^b, c^p, \} \cup fn([\![P]\!])$$

Exactly as before, we can show that the translation from the calculus with channels to the one without preserves typing derivations. This time, however, the translation does not constrain the mobility of processes performing inputs: the mobility effect Z of the process P in a channel input $c(x_1{:}C_1, \ldots, x_k{:}C_k).P$ may be either mobile or immobile.

4 Encoding a Distributed Language

In this section, we consider a fragment of a typed, distributed language in which mobile threads can migrate between immobile network nodes. We obtain a semantics for this form of thread mobility via a translation into the ambient calculus. In the translation, ambients model both threads and

nodes. The translation illustrates the extended type system of Section 3. In particular, the typing of the translation guarantees that an ambient modeling a node moves neither subjectively nor objectively. On the other hand, an ambient modeling a thread is free to move subjectively, but is guaranteed not to move objectively.

The computational model is that there is an unstructured collection of named network nodes, each of which hosts a collection of named communication channels and anonymous threads. This is similar to the computational models underlying various distributed variants of the π-calculus, such as those proposed by Amadio and Prasad [AP94], Riely and Hennessy [RH98], and Sewell [Sew98]. In an earlier paper [CG99], we showed how to mimic Telescript's computational model by translation into the ambient calculus. In the language fragment we describe here, communication is based on named communication channels (as in the π-calculus) rather than by direct agent-to-agent communication (as in our stripped down version of Telescript). As in our previous paper, we focus on language constructs for mobility, synchronization, and communication. We omit standard constructs for data processing and control flow. They could easily be added.

To introduce the syntax of our language fragment, here is a simple example:

$$\begin{array}{l} \mathit{node}\ a\ [\mathit{channel}\ a_c \mid \mathit{thread}[\overline{a_c}\langle b, b_c\rangle]] \mid \mathit{node}\ b\ [\mathit{channel}\ b_c] \mid \\ \mathit{node}\ c\ [\mathit{thread}[\mathit{go}\ a.a_c(x{:}\mathit{Node}, y{:}\mathit{Ch}[\mathit{Node}]).\mathit{go}\ x.\overline{y}\langle a\rangle] \end{array}$$

This program describes a network consisting of three network nodes, named a, b, and c. Node a hosts a channel a_c and a thread running the code $\overline{a_c}\langle b, b_c\rangle$, which simply sends the pair $\langle b, b_c\rangle$ on the channel a_c. Node b hosts a channel b_c. Finally, node c hosts a single thread, running the code:

$$\mathit{go}\ a.a_c(x{:}\mathit{Node}, y{:}\mathit{Ch}[\mathit{Node}]).\mathit{go}\ x.\overline{y}\langle a\rangle$$

The effect of this is to move the thread from node c to node a. There it awaits a message sent on the communication channel a_c. We may assume that it receives the message $\langle b, b_c\rangle$ being sent by the thread already at a. (If there were another thread at node a sending another message, the receiver thread would end up receiving one or other of the messages.) The thread then migrates to node b, where it transmits a message a on the channel b_c.

Messages on communication channels are assigned types, ranged over by W. The type *Node* is the type of names of network nodes. The type

$Ch[W_1, \ldots, W_k]$ is the type of a polyadic communication channel. The messages communicated on such a channel are k-tuples whose components have types $W_1, \ldots, W_k$. In the setting of the example above, channel a_c has type $Ch[Node, Ch[Node]]$, and channel b_c has type $Ch[Node]$.

Next, we describe the formal grammar of our language fragment. A *network*, *Net*, is a collection of nodes, built up using composition $Net \mid Net$ and restrictions $(\nu n{:}W)Net$. A *crowd*, *Cro*, is the group of threads and channels hosted by a node. Like networks, crowds are built up using composition $Cro \mid Cro$ and restriction $(\nu n{:}W)Cro$. A *thread*, *Th*, is a mobile thread of control. As well as the constructs illustrated above, a thread may include the contructs $fork(Cro).Th$ and $spawn\ n\ [Cro].Th$. The first forks a new crowd *Cro* inside the current node, and continues with *Th*. The second spawns a new node $node\ n\ [Cro]$ outside the current node, at the network level, and continues with *Th*.

A fragment of a typed, distributed programming language:

$W ::=$	type
$\quad Node$	$\quad$ name of a node
$\quad Ch[W_1, \ldots, W_k]$	$\quad$ name of a channel
$Net ::=$	network
$\quad (\nu n{:}W)Net$	$\quad$ restriction
$\quad Net \mid Net$	$\quad$ network composition
$\quad node\ n\ [Cro]$	$\quad$ node
$Cro ::=$	crowd of channels and threads
$\quad (\nu n{:}W)Cro$	$\quad$ restriction
$\quad Cro \mid Cro$	$\quad$ crowd composition
$\quad channel\ c$	$\quad$ channel
$\quad thread[Th]$	$\quad$ thread
$Th ::=$	thread
$\quad go\ n.Th$	$\quad$ migration
$\quad \overline{c}\langle n_1, \ldots, n_k \rangle$	$\quad$ output to a channel
$\quad c(x_1{:}W_1, \ldots, x_k{:}W_k).Th$	$\quad$ input from a channel
$\quad fork(Cro).Th$	$\quad$ fork a crowd
$\quad spawn\ n\ [Cro].Th$	$\quad$ spawn a new node

In the phrases $(\nu n{:}W)Net$ and $(\nu n{:}W)Cro$, the name n is bound; its scope is *Net* and *Cro*, respectively. In the phrase $c(x_1{:}W_1, \ldots, x_k{:}W_k).Th$,

the names $x_1, \ldots, x_k$ are bound; their scope is the phrase Th.

The type system of our language controls the typing of messages on communication channels, much as in previous schemes for the π-calculus [Mil91]. We formalize the type system using the following five judgments:

Judgments:

$E \vdash \diamond$	good environment
$E \vdash n : W$	name n has type W
$E \vdash Net$	good network
$E \vdash Cro$	good crowd
$E \vdash Th$	good thread

These judgments are defined by the fairly standard rules in the following tables.

Good environment:

$$\frac{}{\varnothing \vdash \diamond} \qquad \frac{E \vdash \diamond \quad n \notin dom(E)}{E, n{:}W \vdash \diamond}$$

Name has type:

$$\frac{E, n{:}W, E' \vdash \diamond}{E, n{:}W, E' \vdash n : W}$$

Good network:

$$\frac{E, n{:}W \vdash Net}{E \vdash (\nu n{:}W)Net} \qquad \frac{E \vdash Net \quad E \vdash Net'}{E \vdash Net \mid Net'} \qquad \frac{E \vdash n : Node \quad E \vdash Cro}{E \vdash node\ n\,[Cro]}$$

Good crowd:

$$\frac{E, n{:}W \vdash Cro}{E \vdash (\nu n{:}W)\,Cro} \qquad \frac{E \vdash Cro \quad E \vdash Cro'}{E \vdash Cro \mid Cro'}$$

$$\frac{E \vdash c : Ch[W_1, \ldots, W_k]}{E \vdash channel\ c} \qquad \frac{E \vdash Th}{E \vdash thread[Th]}$$

Good thread:

$$\frac{E \vdash n : \mathit{Node} \quad E \vdash \mathit{Th}}{E \vdash \mathit{go}\ n.\mathit{Th}}$$

$$\frac{E \vdash c : \mathit{Ch}[W_1, \ldots, W_k] \quad E \vdash n_i : W_i \quad \forall i \in 1..k}{E \vdash \overline{c}\langle n_1, \ldots, n_k \rangle}$$

$$\frac{E \vdash c : \mathit{Ch}[W_1, \ldots, W_k] \quad E, x_1{:}W_1, \ldots, x_k{:}W_k \vdash \mathit{Th}}{E \vdash c(x_1{:}W_1, \ldots, x_k{:}W_k).\mathit{Th}}$$

$$\frac{E \vdash \mathit{Cro} \quad E \vdash \mathit{Th}}{E \vdash \mathit{fork}(\mathit{Cro}).\mathit{Th}} \qquad \frac{E \vdash n : \mathit{Node} \quad E \vdash \mathit{Cro} \quad E \vdash \mathit{Th}}{E \vdash \mathit{spawn}\ n\,[\mathit{Cro}].\mathit{Th}}$$

We give a semantics to this language fragment via a translation into the ambient calculus. In the semantics, we use ambients to model the three notions of node, thread, and channel.

We model a node named n by a box ambient named n^b, that is, n^b is a box holding the contents of the node. We assign n the type $\mathit{Amb}^{\bullet}\!\curvearrowright\![\overset{\vee}{-}\mathit{Shh}]$, that is, it is locked, subjectively immobile, but objectively mobile. In our semantics, we never need to open up or subjectively move such an ambient, but we need to use an objective move to model a thread spawning a new node. To assist in the translation of threads that fork crowds and spawn nodes, we situate a replicated process $\mathit{open}\ n^p$ within each ambient that models a node n. The auxiliary name n^p, which is derived from the name n, is assigned the type $\mathit{Amb}^{\circ}\!\curvearrowright\![\overset{\vee}{-}\mathit{Shh}]$.

Much as in Section 3, we model a channel named c by a buffer ambient named c^b, and we model a packet sent on the channel by a packet ambient named c^p. We assign the types $\mathit{Amb}^{\bullet\overset{\vee}{-}}[\overset{\vee}{-}T]$ and $\mathit{Amb}^{\circ}\!\curvearrowright\![\overset{\vee}{-}T]$, respectively, to these ambients, where T is the type of tuples carried by the channel.

We model a thread by an ambient. We assign ambients modeling threads the type $\mathit{Amb}^{\bullet\overset{\vee}{-}}[\curvearrowright\mathit{Shh}]$. We use subjective moves to model thread migration. We never dissolve threads nor use objective moves on a thread.

Given these explanations, we now present the translation of the language fragment into our calculus. We begin by translating the type W of a node or a channel into an ambient type $[\![W]\!]$.

Translation of a type:

$$
\begin{array}{l}
[\![Node]\!]^b \triangleq Amb^{\bullet}\!\curvearrowright\![\veebar Shh] \\
[\![Node]\!]^p \triangleq Amb^{\circ}\!\curvearrowright\![\veebar Shh] \\
[\![Ch[W_1,\ldots,W_k]]\!]^b \triangleq Amb^{\bullet\veebar}\![\veebar [\![W_1]\!]^b \times [\![W_1]\!]^p \times \cdots \times [\![W_k]\!]^b \times [\![W_k]\!]^p] \\
[\![Ch[W_1,\ldots,W_k]]\!]^p \triangleq Amb^{\circ}\!\curvearrowright\![\veebar [\![W_1]\!]^b \times [\![W_1]\!]^p \times \cdots \times [\![W_k]\!]^b \times [\![W_k]\!]^p]
\end{array}
$$

Now, we translate a network Net, a crowd Grp within a node n, and a thread Th within a node n and assigned a name t, to the ambient processes $[\![Net]\!]$, $[\![Grp]\!]_n$, and $[\![Th]\!]^t_n$, respectively.

Translation of a network:

$$
\begin{array}{l}
[\![(\nu n{:}W)\,Net]\!] \triangleq (\nu n^b{:}[\![W]\!]^b)(\nu n^p{:}[\![W]\!]^p)[\![Net]\!] \\
[\![Net \mid Net]\!] \triangleq [\![Net]\!] \mid [\![Net]\!] \\
[\![node\ n\ [Cro]]\!] \triangleq n^b[!open\ n^p \mid [\![Cro]\!]_n]
\end{array}
$$

Translation of a crowd Cro located at n:

$$
\begin{array}{l}
[\![(\nu c{:}W)\,Cro]\!]_n \triangleq (\nu c^b{:}[\![W]\!]^b)(\nu c^p{:}[\![W]\!]^p)[\![Cro]\!]_n \\
[\![Cro \mid Cro]\!]_n \triangleq [\![Cro]\!]_n \mid [\![Cro]\!]_n \\
[\![channel\ c]\!]_n \triangleq c^b[!open\ c^p] \\
[\![thread\ Th]\!]_n \triangleq (\nu t{:}Amb^{\bullet\veebar}\![\curvearrowright Shh])t[[\![Th]\!]^t_n] \quad \text{for } t \notin fn([\![Th]\!]^t_n)
\end{array}
$$

Translation of a thread Th named t located at n:

$$
\begin{array}{l}
[\![go\ m.Th]\!]^t_n \triangleq out\ n.in\ m.[\![Th]\!]^t_m \\
[\![\overline{c}\langle n_1,\ldots,n_k\rangle]\!]^t_n \triangleq go\ (out\ t.in\ c^b).c^p[\langle n_1, n_1^p, \ldots, n_k, n_k^p\rangle] \\
[\![c(x_1{:}W_1,\ldots,x_k{:}W_k).Th]\!]^t_n \triangleq \\
\quad (\nu s{:}Amb^{\circ}\!\curvearrowright\![\curvearrowright Shh]) \\
\quad\quad (go\ (out\ t.in\ c^b). \\
\quad\quad\quad c^p[(x_1^b{:}[\![W_1]\!]^b, x_1^p{:}[\![W_1]\!]^p, \ldots, x_k^b{:}[\![W_k]\!]^b, x_k^p{:}[\![W_k]\!]^p). \\
\quad\quad\quad\quad go\ (out\ c^b.in\ t).s[open\ s.[\![Th]\!]^t_n]] \mid \\
\quad\quad\quad open\ s.s[]) \\
\quad \text{for } s \notin \{t, c^b, c^p\} \cup fn([\![Th]\!]^t_n)
\end{array}
$$

$[\![fork(Cro).Th]\!]^t_n \triangleq$
$\quad (\nu s{:}Amb^{\circ\frown}[\frown Shh])$
$\quad\quad (go\ out\ t.n^p[go\ (in\ t).s[] \mid [\![Cro]\!]_n] \mid$
$\quad\quad\ open\ s.[\![Th]\!]^t_n)$
$\quad$ for $s \notin \{t, n^p\} \cup [\![Cro]\!]_n \cup [\![Th]\!]^t_n$
$[\![spawn\ m\ [Cro].Th]\!]^t_n \triangleq$
$\quad (\nu s{:}Amb^{\circ\frown}[\frown Shh])$
$\quad\quad (go\ out\ t.n^p[go\ in\ t.s[] \mid$
$\quad\quad\quad\quad go\ out\ n^b.m^b[!open\ m^p \mid [\![Cro]\!]_m]] \mid$
$\quad\quad\ open\ s.[\![Th]\!]^t_n)$
$\quad$ for $s \notin \{t, n^b, n^p, m^b, m^p\} \cup fn([\![Cro]\!]_m) \cup fn([\![Th]\!]^t_n)$

We translate an input operation $c(x_1{:}W_1, ..., x_k{:}W_k).Th$ as a packet c^p which exits the enclosing t ambient and then enters the channel c^b. There it is opened, and its input operation awaits a message. After the input receives a message, the continuation $s[open\ s.[\![Th]\!]^t_n]$ exits the channel and re-enters the thread t. Here it encounters the process $open\ s.s[]$. The $open\ s$ capability dissolves the first s ambient, the one used to move the continuation back, but unleashes a second s ambient. At this point, the $open\ s$ which has been guarding the $[\![Th]\!]^t_n$ process can open the second s ambient, and the $[\![Th]\!]^t_n$ can proceed. We need to guard the $[\![Th]\!]^t_n$ process to prevent a subjective move within $[\![Th]\!]^t_n$ from acting on the first s ambient instead of the t ambient. We translate a *fork* operation as a packet n^p which carries the new crowd *Cro* out of its enclosing thread to become a child of the enclosing node n^b. Once the packet n^p is a child of n, it is opened, the translation $[\![Cro]\!]_n$ of the crowd *Cro* can run, and the synchronization packet $s[]$ re-enters the t ambient where it is opened by the $open s$ lock which was guarding $[\![Th]\!]^t_n$. This synchronization is needed to avoid the process $[\![Th]\!]^t_n$ moving the ambient t to another node before the new crowd has exited from t. The translation of the *spawn* operation is similar.

In order to state Proposition 2, which asserts that our translation preserves typing judgments, we use the following translation of an environment E of the language fragment to an environment $[\![E]\!]$ of the ambient calculus.

Translation of an environment:

$[\![\varnothing]\!] \triangleq \varnothing$
$[\![E, n{:}W]\!] \triangleq [\![E]\!], n^b{:}[\![W]\!]^b, n^p{:}[\![W]\!]^p$

Proposition 2

(1) *If* $E \vdash Net$ *then* $[\![E]\!] \vdash [\![Net]\!] : {}^{\vee}_{-}Shh$.

(2) *If* $E \vdash Cro$ *and* $E \vdash n : Node$ *then* $[\![E]\!] \vdash [\![Cro]\!]_n : {}^{\vee}_{-}Shh$.

(3) *If* $E \vdash Th$, $E \vdash n : Node$, $t \notin dom(E)$ *then* $[\![E]\!], t{:}Amb^{\bullet}{}^{\vee}_{-}[\curvearrowright Shh] \vdash [\![Th]\!]^t_n : \curvearrowright Shh$.

Proof By induction on derivations. □

This example shows that the ambient calculus serves as a semantic metalanguage for describing distributed computation. Ambients model several different entities, including nodes, channels, and threads. The different typings for these entities reflect their different behavior. Of the three kinds of ambients, only nodes allow objective moves, only channels allow message exchange, and only threads allow subjective moves. The example shows that our mobility and locking annotations are useful for the mobile programming task underlying this translation.

5 Conclusions and Related Work

We have argued [CG98, Car99, CG99, GC99] that the idea of an ambient is a useful and general abstraction for expressing and reasoning about mobile computation. In this paper, we qualified the ambient idea by introducing type systems that distinguish between mobile and immobile, and locked and unlocked ambients. Thus qualified, ambients better describe the structure of mobile computations.

The type systems presented in this paper derive from our earlier work on exchange types for ambients [CG99]. That type system tracks the types of messages that may be input or output within each ambient; it is analogous to Milner's sort system for the π-calculus [Mil91], which tracks the types of messages that may be input or output on each channel.

Our mobility annotations govern the ways in which an ambient can be moved. The data movement types of the mobile λ-calculus of Sekiguchi and Yonezawa [SY97] also govern movement, the movement of variables referred to by mobile processes. Their data movement types are checked dynamically, rather than statically. In the setting of the π-calculus, various type systems

have been proposed to track the distinction between local and remote references to channels [Ama97, Sew98, SWP98], but none of these systems tracks process mobility.

Our locking annotations allow static checking of a simple security property: that nobody will attempt to open a locked ambient. More complex type systems than ours demonstrate that more sophisticated security properties of concurrent systems can be checked statically: access control [DFPV98, HR98b], allocation of permissions [RH98], and secrecy and integrity properties [Aba97, HR98a, SV98]. Ideas from some of these systems may be applicable to ambients.

Moreover, for the sake of programming convenience, our type systems could be extended in standard directions, just as Milner's sort system for the π-calculus has been extended with subtyping [PS96] and parametric polymorphism [Tur95]. More experimental extensions to the π-calculus, such as affine or linear types [KPT96] and graph types [Yos96], may also be useful extensions to the type systems of this paper. In earlier work [CG99], we studied the addition of affine capability types to our basic system of exchange types.

Acknowledgement

Giorgio Ghelli acknowledges the support of Microsoft Research during the writing of this paper. This work has also been partially supported by Esprit Working Groups 26142 - Applied Semantics and 22552 - PASTEL, and by Italian MURST, project InterData.

References

[Aba97] M. Abadi. Secrecy by typing in security protocols. In *Proceedings TACS'97*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–638. Springer, 1997.

[Ama97] R. M. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proceedings COORDINATION 97*, volume 1282 of *Lecture Notes in Computer Science*. Springer, 1997.

[AP94] R. M. Amadio and S. Prasad. Localities and failures. In *Proceedings FST&TCS'94*, volume 880 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 1994.

[Car99] L. Cardelli. Abstractions for mobile computation. In C. Jensen and J. Vitek, editors, *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, volume 1603 of *Lecture Notes in Computer Science*. Springer, 1999.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceedings FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[CG99] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Proceedings POPL'99*, pages 79–92. ACM, January 1999.

[CGG99] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. Unpublished, 1999.

[DFPV98] R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. Available from the authors, 1998.

[GC99] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *Proceedings FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 1999. An extended version appears as Microsoft Research Technical Report MSR–TR–99–11, April 1999.

[HR98a] N. Heintz and J. Riecke. The SLam calculus: programming with secrecy and integrity. In *Proceedings POPL'98*, pages 365–377. ACM, 1998.

[HR98b] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In *Proceedings HLCL'98*, volume 16(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998.

[KPT96] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. In *Proceedings POPL'96*, pages 358–371. ACM, 1996.

[LY97] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1997.

[Mil91] R. Milner. The polyadic π-calculus: A tutorial. Technical Report ECS–LFCS–91–180, University of Edinburgh, October 1991.

[PS96] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[RH98] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proceedings POPL'98*, pages 378–390. ACM, 1998.

[Sew98] P. Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proceedings ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 695–706. Springer, 1998.

[SV98] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings POPL'98*, pages 355–364. ACM, 1998.

[SWP98] P. Sewell, P. T. Wojciechowski, and B.C. Pierce. Location independence for mobile agents. In *Workshop on Internet Programming Languages*, 1998.

[SY97] T. Sekiguchi and A. Yonezawa. A calculus with code mobility. In *Proceedings FMOODS'97*, pages 21–36. IFIP, 1997.

[Tur95] D. N. Turner. *The polymorphic pi-calculus: theory and implementation.* PhD thesis, University of Edinburgh, 1995.

[Yos96] N. Yoshida. Graph types for monadic mobile processes. In *Proceeings FST&TCS*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer, 1996.

A Proof of Theorem 1

Lemma 3 (Derived Judgment) *If $E \vdash \mathcal{J}$ then $E \vdash \diamond$.*

Lemma 4 *If $E', n{:}W, E'' \vdash \mathcal{J}$ then $n \notin dom(E', E'')$.*

Lemma 5 *If $E \vdash n : W$ and $E \vdash n : W'$ then $W = W'$.*

Lemma 6 (Exchange)
If $E', n{:}W', m{:}W'', E'' \vdash \mathcal{J}$ then $E', m{:}W'', n{:}W', E'' \vdash \mathcal{J}$.

Lemma 7 (Strengthening)
If $E', n{:}W, E'' \vdash \mathcal{J}$ and $n \notin fn(\mathcal{J})$ then $E', E'' \vdash \mathcal{J}$.

Lemma 8 (Weakening)
If $E', E'' \vdash \mathcal{J}$ and $n \notin dom(E', E'')$ then $E', n{:}W, E'' \vdash \mathcal{J}$.

Lemma 9 (Substitution)
If $E', n{:}W, E'' \vdash \mathcal{J}$ and $E' \vdash M : W$ then $E', E'' \vdash \mathcal{J}\{n \leftarrow M\}$.

Proposition 10 *If $E \vdash P : {}^{Z}T$ and $P \equiv Q$ then $E \vdash Q : {}^{Z}T$.*

Proof The proposition follows by showing that $P \equiv Q$ implies:

(1) If $E \vdash P : {}^{Z}T$ then $E \vdash Q : {}^{Z}T$.

(2) If $E \vdash Q : {}^{Z}T$ then $E \vdash P : {}^{Z}T$.

We proceed by induction on the derivation of $P \equiv Q$.

(Struct Refl) Trivial.

(Struct Symm) Then $Q \equiv P$. For (1), assume $E \vdash P : {}^{Z}T$. By induction hypothesis (2), $Q \equiv P$ implies $E \vdash Q : {}^{Z}T$. Part (2) is symmetric.

(Struct Trans) Then $P \equiv R$, $R \equiv Q$ for some R. For (1), assume $E \vdash P : {}^{Z}T$. By induction hypothesis (1), $E \vdash R : {}^{Z}T$. Again by induction hypothesis (1), $E \vdash Q : {}^{Z}T$. Part (2) is symmetric.

(Struct Res) Then $P = (\nu n{:}W)P'$ and $Q = (\nu n{:}W)Q'$, with $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res), with $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash P' : {}^{Z}T$, where $W = Amb^{Y'}[{}^{Z'}T']$. By induction hypothesis (1), $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash Q' : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}W)Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Par) Then $P = P' \mid R$, $Q = Q' \mid R$, and $P' \equiv Q'$. For (1), assume $E \vdash P' \mid R : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash P' : {}^{Z}T$, $E \vdash R : {}^{Z}T$. By induction hypothesis (1), $E \vdash Q' : {}^{Z}T$. By (Proc Par), $E \vdash Q' \mid R : {}^{Z}T$. Part (2) is symmetric.

(Struct Repl) Then $P = !P'$, $Q = !Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Repl), with $E \vdash P' : {}^{Z}T$. By induction hypothesis (1), $E \vdash Q' : {}^{Z}T$. By (Proc Repl), $E \vdash !Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Amb) Then $P = M[P']$, $Q = M[Q']$, and $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Amb), with $E \vdash M : Amb^{Y'}[{}^{Z'}T']$ and $E \vdash P' : {}^{Z'}T'$, for some Y', Z', T'. By induction hypothesis (1), $E \vdash Q' : {}^{Z'}T'$. By (Proc Amb), $E \vdash M[Q'] : {}^{Z}T$. Part (2) is symmetric.

(Struct Action) Then $P = M.P'$, $Q = M.Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Action), with $E \vdash M : Cap[{}^{Z}T]$, $E \vdash P' : {}^{Z}T$. By induction hypothesis (1), $E \vdash Q' : {}^{Z}T$. By (Proc Action), $E \vdash M.Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Input) In this case, $P = (n_1{:}W_1, \ldots, n_k{:}W_k).P'$, $P' \equiv Q'$, and $Q = (n_1{:}W_1, \ldots, n_k{:}W_k).Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Input), with $E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P' : {}^{Z}T$, where $T = W_1 \times \cdots \times W_k$. By induction hypothesis, we have $E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash Q' : {}^{Z}T$. By (Proc Input), we have $E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Par Comm) Then $P = P' \mid P''$ and $Q = P'' \mid P'$.

For (1), assume $E \vdash P' \mid P'' : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash P' : {}^{Z}T$ and $E \vdash P'' : {}^{Z}T$. By (Proc Par), $E \vdash P'' \mid P' : {}^{Z}T$. Hence, $E \vdash Q : {}^{Z}T$.

Part (2) is symmetric.

(Struct Par Assoc) Then $P = (P' \mid P'') \mid P'''$ and $Q = P' \mid (P'' \mid P''')$.

For (1), assume $E \vdash (P' \mid P'') \mid P''' : {}^{Z}T$. This must have been derived from (Proc Par) twice, with $E \vdash P' : {}^{Z}T$, $E \vdash P'' : {}^{Z}T$, and $E \vdash P''' : {}^{Z}T$. By (Proc Par) twice, $E \vdash P' \mid (P'' \mid P''') : {}^{Z}T$. Hence $E \vdash Q : {}^{Z}T$.

Part (2) is symmetric.

(Struct Repl Par) Then $P = !P'$ and $Q = P' \mid !P'$.

For (1), assume $E \vdash !P' : {}^{Z}T$. This must have been derived from (Proc Repl), with $E \vdash P' : {}^{Z}T$. By (Proc Par), $E \vdash P' \mid !P' : {}^{Z}T$. Hence, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash P' \mid !P' : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash P' : {}^{Z}T$ and $E \vdash !P' : {}^{Z}T$. Hence, $E \vdash P : {}^{Z}T$.

(Struct Res Res) In this case, we have $P = (\nu n_1{:}W_1)(\nu n_2{:}W_2)P'$ and $Q = (\nu n_2{:}W_2)(\nu n_1{:}W_1)P'$ with $n_1 \neq n_2$.

For (1), assume $E \vdash (\nu n_1{:}W_1)(\nu n_2{:}W_2)P' : {}^{Z}T$. This must have been derived from (Proc Res), with $E, n_1{:}Amb^{Y_1}[{}^{Z_1}T_1], n_2{:}Amb^{Y_2}[{}^{Z_2}T_2] \vdash P' : {}^{Z}T$, where $W_1 = Amb^{Y_1}[{}^{Z_1}T_1]$ and $W_2 = Amb^{Y_2}[{}^{Z_2}T_2]$. By Lemma 6, we have $E, n_2{:}Amb^{Y_2}[{}^{Z_2}T_2], n_1{:}Amb^{Y_1}[{}^{Z_1}T_1] \vdash P' : {}^{Z}T$. By (Proc Res) twice we have $E \vdash (\nu n_2{:}W_2)(\nu n_1{:}W_1)P' : {}^{Z}T$. Part (2) is symmetric.

(Struct Res Par) Then $P = (\nu n{:}W)(P' \mid P'')$ and $Q = P' \mid (\nu n{:}W)P''$, with $n \notin fn(P')$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res), with $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash P' \mid P'' : {}^{Z}T$ and $W = Amb^{Y'}[{}^{Z'}T']$, and (Proc Par), with $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash P' : {}^{Z}T$ and $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash P'' : {}^{Z}T$. By Lemma 7, since $n \notin fn(P')$, we have $E \vdash P' : {}^{Z}T$. By (Proc Res) we have $E \vdash (\nu n{:}Amb^{Y'}[{}^{Z'}T'])P'' : {}^{Z}T$. By (Proc Par) we have $E \vdash P' \mid (\nu n{:}Amb^{Y'}[{}^{Z'}T'])P'' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash Q : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash P' : {}^{Z}T$ and $E \vdash (\nu n{:}W)P'' : {}^{Z}T$, and from (Proc Res), with $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash P'' : {}^{Z}T$ and $W = Amb^{Y'}[{}^{Z'}T']$. By Lemma 4, $n \notin dom(E)$. By Lemma 8, $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash P' : {}^{Z}T$. By (Proc Par), $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash P' \mid P'' : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}Amb^{Y'}[{}^{Z'}T'])(P' \mid P'') : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Res Amb) Then $P = (\nu n{:}W)m[P']$ and $Q = m[(\nu n{:}W)P']$, with $n \neq m$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res) with $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash m[P'] : {}^{Z}T$ with $W = Amb^{Y'}[{}^{Z'}T']$, and from (Proc Amb) with $E, n{:}Amb^{Y'}[{}^{Z'}T'] \vdash m : Amb^{Y_m}[{}^{Z_m}T_m]$ and

$E, n{:}Amb^{Y'}[^{Z'}T'] \vdash P' : {}^{Z_m}T_m$ for some Y_m, Z_m, T_m. By (Proc Res) we have $E \vdash (\nu n{:}Amb^{Y'}[^{Z'}T'])P' : {}^{Z_m}T_m$. By Lemma 7, $n \neq m$ implies $E \vdash m : Amb^{Y_m}[^{Z_m}T_m]$. By (Proc Amb), $E \vdash m[(\nu n{:}Amb^{Y'}[^{Z'}T'])P'] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash Q : {}^{Z}T$. This must have been derived from (Proc Amb) with $E \vdash m : Amb^{Y_m}[^{Z_m}T_m]$ and $E \vdash (\nu n{:}W)P' : {}^{Z_m}T_m$, and from (Proc Res), with $E, n{:}Amb^{Y'}[^{Z'}T'] \vdash P' : {}^{Z_m}T_m$ and $W = Amb^{Y'}[^{Z'}T']$, for some Y_m, Z_m, and T_m. By Lemma 4, $n \notin dom(E)$. By Lemma 8, $E, n{:}Amb^{Y'}[^{Z'}T'] \vdash m : Amb^{Y_m}[^{Z_m}T_m]$. By (Proc Amb), $E, n{:}Amb^{Y'}[^{Z'}T'] \vdash m[P'] : {}^{Z}T$. By (Proc Res), we can derive that $E \vdash (\nu n{:}Amb^{Y'}[^{Z'}T'])m[P'] : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Zero Par) Then $P = P' \mid \mathbf{0}$ and $Q = P'$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Par) with $E \vdash P' : {}^{Z}T$ and $E \vdash \mathbf{0} : {}^{Z}T$. Hence, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash P' : {}^{Z}T$. By Lemma 3, $E \vdash \diamond$. By (Proc Zero), $E \vdash \mathbf{0} : {}^{Z}T$. By (Proc Par), $E \vdash P' \mid \mathbf{0} : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Zero Res) Then $P = (\nu n{:}Amb^{Y'}[^{Z'}T'])\mathbf{0}$ and $Q = \mathbf{0}$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res) with $E, n{:}Amb^{Y'}[^{Z'}T'] \vdash \mathbf{0} : {}^{Z}T$. By Lemma 7, $E \vdash \mathbf{0} : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash \mathbf{0} : {}^{Z}T$. We may assume that the bound name n does not occur in $dom(E)$. By Lemma 8, $E, n{:}Amb^{Y'}[^{Z'}T'] \vdash \mathbf{0} : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}Amb^{Y'}[^{Z'}T'])\mathbf{0} : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Zero Repl) Then $P = !\mathbf{0}$ and $Q = \mathbf{0}$.

For (1), assume $E \vdash P : {}^{Z}T$. By Lemma 3, $E \vdash \diamond$. By (Proc Zero), $E \vdash \mathbf{0} : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash \mathbf{0} : {}^{Z}T$. By (Proc Repl), $E \vdash !\mathbf{0} : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct ϵ) Then $P = \epsilon.P'$ and $Q = P'$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Action) with $E \vdash \epsilon : Cap[^{Z}T]$ and $E \vdash P' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash P' : {}^{Z}T$. By Lemma 3, $E \vdash \diamond$. By (Exp ϵ), $E \vdash \epsilon : Cap[^{Z}T]$. By (Proc Action), $E \vdash \epsilon.P' : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct .) Then $P = (M.M').P'$ and $Q = M.M'.P'$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Action) with $E \vdash P' : {}^{Z}T$, $E \vdash M.M' : Cap[{}^{Z}T]$. The latter must have come from (Exp .) with $E \vdash M : Cap[{}^{Z}T]$ and $E \vdash M' : Cap[{}^{Z}T]$, By (Proc Action) twice, $E \vdash M.(M'.P') : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash Q : {}^{Z}T$. This must have been derived from (Proc Action), twice, with $E \vdash M : Cap[{}^{Z}T]$, $E \vdash M' : Cap[{}^{Z}T]$, and $E \vdash P' : {}^{Z}T$. By (Exp .), $E \vdash M.M' : Cap[{}^{Z}T]$. By (Proc Action), $E \vdash (M.M').P' : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$. □

Proof of Theorem 1 *If $E \vdash P : {}^{Z}T$ and $P \to Q$ then $E \vdash Q : {}^{Z}T$.*

Proof By induction on the derivation of $P \to Q$.

(Red In) Then $P = n[in\ m.P' \mid P''] \mid m[P''']$ and $Q = m[n[P' \mid P''] \mid P''']$. Assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash n[in\ m.P' \mid P''] : {}^{Z}T$ and $E \vdash m[P'''] : {}^{Z}T$. The former must have been derived from (Proc Amb), with $E \vdash n : Amb^{Y_n}[{}^{Z_n}T_n]$ and $E \vdash in\ m.P' \mid P'' : {}^{Z_n}T_n$, for some Y_n, Z_n, and T_n, while the latter must have been derived from (Proc Amb) with $E \vdash m : Amb^{Y_m}[{}^{Z_m}T_m]$ and $E \vdash P''' : {}^{Z_m}T_m$, for some Y_m, Z_m, and T_m. Moreover, $E \vdash inm.P' \mid P'' : {}^{Z_n}T_n$ must come from (Proc Par) with $E \vdash inm.P' : {}^{Z_n}T_n$ and $E \vdash P'' : {}^{Z_n}T_n$. Finally, $E \vdash inm.P' : {}^{Z_n}T_n$ must come from $E \vdash inm : Cap[{}^{Z_n}T_n]$ and $E \vdash P' : {}^{Z_n}T_n$. (Hence $Z_n = \curvearrowright$, since the former judgment can only be derived using (Exp In).) By (Proc Par), we have $E \vdash P' \mid P'' : {}^{Z_n}T_n$, and by (Proc Amb) we can derive $E \vdash n[P' \mid P''] : {}^{Z_m}T_m$. Then, by (Proc Par), we have $E \vdash n[P' \mid P''] \mid P''' : {}^{Z_m}T_m$. By (Proc Amb) we can derive $E \vdash m[n[P' \mid P''] \mid P'''] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Out) Then $P = m[n[out\ m.P' \mid P''] \mid P'''$ and $Q = n[P' \mid P''] \mid m[P''']$. Assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Amb) using $E \vdash m : Amb^{Y_m}[{}^{Z_m}T_m]$ and $E \vdash n[out\ m.P' \mid P''] \mid P''' : {}^{Z_m}T_m$ for some Y_m, Z_m, and T_m, and from (Proc Par) using $E \vdash n[out\ m.P' \mid P''] : {}^{Z_m}T_m$ and $E \vdash P''' : {}^{Z_m}T_m$. The former must have been derived using (Proc Amb) from $E \vdash n : Amb^{Y_n}[{}^{Z_n}T_n]$ and $E \vdash out\ m.P' \mid P'' : {}^{Z_n}T_n$ for some Y_n, Z_n, and T_n, and using (Proc Par) from $E \vdash out\ m.P' : {}^{Z_n}T_n$ and $E \vdash P'' : {}^{Z_n}T_n$. The former must have been derived using (Proc Action) from $E \vdash out\ m : Cap[{}^{Z_n}T_n]$

and $E \vdash P' : {}^{Z_n}T_n$ (Hence $Z_n = \curvearrowright$, since the former judgment can only be derived using (Exp Out).) By (Proc Par), $E \vdash P' \mid P'' : {}^{Z_n}T_n$. By (Proc Amb), $E \vdash n[P' \mid P''] : {}^{Z}T$. By (Proc Amb), $E \vdash m[P'''] : {}^{Z_n}T_n$. By (Proc Par), $E \vdash n[P' \mid P''] \mid m[P'''] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Open) Then $P = \mathit{open}\ n.P' \mid n[P'']$ and $Q = P' \mid P''$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Par) from $E \vdash \mathit{open}\ n.P' : {}^{Z}T$ and $E \vdash n[P''] : {}^{Z}T$. The former must have been derived using (Proc Action) with $E \vdash \mathit{open}\ n : \mathit{Cap}[{}^{Z}T]$ and $E \vdash P' : {}^{Z}T$, while the latter must have been derived using (Proc Amb) with $E \vdash n : \mathit{Amb}^{Y'}[{}^{Z'}T']$ and $E \vdash P'' : {}^{Z'}T'$ for some Y', Z', and T'. The judgment $E \vdash \mathit{open}\ n : \mathit{Cap}[{}^{Z}T]$ must have been derived using (Exp Open) from $E \vdash n : \mathit{Amb}^{\circ}[{}^{Z}T]$. By Lemma 5, $\mathit{Amb}^{Y'}[{}^{Z'}T'] = \mathit{Amb}^{\circ}[{}^{Z}T]$, and so $Y' = \circ$, $Z' = Z$ and $T' = T$. Hence, $E \vdash P'' : {}^{Z}T$, and, by (Proc Par), $E \vdash P' \mid P'' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Comm) Then $P = (n_1{:}W_1, \ldots, n_k{:}W_k).P' \mid \langle M_1, \ldots, M_k \rangle$ and $Q = P'\{n_1 \leftarrow M_1, \ldots, n_k \leftarrow M_k\}$. Assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Par) with $E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P' : {}^{Z}T$ and $E \vdash \langle M_1, \ldots, M_k \rangle : {}^{Z}T$. The former must have been derived from (Proc Input) with $E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P' : {}^{Z}T$ and $T = W_1 \times \cdots W_k$. The latter judgment $E \vdash \langle M_1, \ldots, M_k \rangle : {}^{Z}T$ must have been derived from (Proc Output) with $E \vdash M_i : W_i'$ for each $i \in 1..k$, and $T = W_1' \times \cdots \times W_k'$. Hence $W_i' = W_i$ for each $i \in 1..k$. By k applications of Lemma 9, we get $E \vdash P'\{n_1 \leftarrow M_1, \ldots, n_k \leftarrow M_k\} : {}^{Z}T$.

(Red Res) Here $P = (\nu n{:}W)P'$ and $Q = (\nu n{:}W)Q'$ with $P' \to Q'$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Res) from $E, n{:}\mathit{Amb}^{Y'}[{}^{Z'}T'] \vdash P' : {}^{Z}T$ with $W = \mathit{Amb}^{Y'}[{}^{Z'}T']$. By induction hypothesis, $E, n{:}\mathit{Amb}^{Y'}[{}^{Z'}T'] \vdash Q' : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}\mathit{Amb}^{Y'}[{}^{Z'}T'])Q' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Amb) Here $P = n[P']$ and $Q = n[Q']$ with $P' \to Q'$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Amb) from $E \vdash n : \mathit{Amb}^{Y'}[{}^{Z'}T']$ and $E \vdash P' : {}^{Z'}T'$. By induction hypothesis, $E \vdash Q' : {}^{Z'}T'$. By (Proc Amb), $E \vdash n[Q'] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Par) Here $P = P' \mid R$ and $Q = Q' \mid R$ with $P' \to Q'$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Par) from

$E \vdash P' : {}^{Z}T$ and $E \vdash R : {}^{Z}T$. By induction hypothesis, $E \vdash Q' : {}^{Z}T$. By (Proc Par), $E \vdash Q' \mid R : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red $\equiv$) Here $P \equiv P'$, $P' \to Q'$, and $Q' \equiv Q$. Assume $E \vdash P : {}^{Z}T$. By Proposition 10, $E \vdash P' : {}^{Z}T$. By induction hypothesis, $E \vdash Q' : {}^{Z}T$. By Proposition 10, $E \vdash Q : {}^{Z}T$. □

B Proof of Theorem 2

Lemma 11 (Derived Judgment) *If $E \vdash \mathcal{J}$ then $E \vdash \diamond$.*

Lemma 12 *If $E', n{:}W, E'' \vdash \mathcal{J}$ then $n \notin dom(E', E'')$.*

Lemma 13 *If $E \vdash n : W$ and $E \vdash n : W'$ then $W = W'$.*

Lemma 14 (Exchange)
If $E', n{:}W', m{:}W'', E'' \vdash \mathcal{J}$ then $E', m{:}W'', n{:}W', E'' \vdash \mathcal{J}$.

Lemma 15 (Strengthening)
If $E', n{:}W, E'' \vdash \mathcal{J}$ and $n \notin fn(\mathcal{J})$ then $E', E'' \vdash \mathcal{J}$.

Lemma 16 (Weakening)
If $E', E'' \vdash \mathcal{J}$ and $n \notin dom(E', E'')$ then $E', n{:}W, E'' \vdash \mathcal{J}$.

Lemma 17 (Substitution)
If $E', n{:}W, E'' \vdash \mathcal{J}$ and $E' \vdash M : W$ then $E', E'' \vdash \mathcal{J}\{n{\leftarrow}M\}$.

Proposition 18 *If $E \vdash P : {}^{Z}T$ and $P \equiv Q$ then $E \vdash Q : {}^{Z}T$.*

Proof The proposition follows by showing that $P \equiv Q$ implies:

(1) If $E \vdash P : {}^{Z}T$ then $E \vdash Q : {}^{Z}T$.

(2) If $E \vdash Q : {}^{Z}T$ then $E \vdash P : {}^{Z}T$.

We proceed by induction on the derivation of $P \equiv Q$.

(Struct Refl) Trivial.

(Struct Symm) Then $Q \equiv P$. For (1), assume $E \vdash P : {}^{Z}T$. By induction hypothesis (2), $Q \equiv P$ implies $E \vdash Q : {}^{Z}T$. Part (2) is symmetric.

(Struct Trans) Then $P \equiv R$, $R \equiv Q$ for some R. For (1), assume $E \vdash P : {}^{Z}T$. By induction hypothesis (1), $E \vdash R : {}^{Z}T$. Again by induction hypothesis (1), $E \vdash Q : {}^{Z}T$. Part (2) is symmetric.

(Struct Res) Then $P = (\nu n{:}W)P'$ and $Q = (\nu n{:}W)Q'$, with $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res), with $E, n{:}Amb^{Y'\,Z''}[{}^{Z'}T'] \vdash P' : {}^{Z}T$, where $W = Amb^{Y'\,Z''}[{}^{Z'}T']$. By induction hypothesis (1), $E, n{:}Amb^{Y'\,Z''}[{}^{Z'}T'] \vdash Q' : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}W)Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Par) Then $P = P' \mid R$, $Q = Q' \mid R$, and $P' \equiv Q'$. For (1), assume $E \vdash P' \mid R : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash P' : {}^{Z}T$, $E \vdash R : {}^{Z}T$. By induction hypothesis (1), $E \vdash Q' : {}^{Z}T$. By (Proc Par), $E \vdash Q' \mid R : {}^{Z}T$. Part (2) is symmetric.

(Struct Repl) Then $P = !P'$, $Q = !Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Repl), with $E \vdash P' : {}^{Z}T$. By induction hypothesis (1), $E \vdash Q' : {}^{Z}T$. By (Proc Repl), $E \vdash !Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Amb) Then $P = M[P']$, $Q = M[Q']$, and $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Amb), with $E \vdash M : Amb^{Y'\,Z''}[{}^{Z'}T']$ and $E \vdash P' : {}^{Z'}T'$, for some Y', Z', Z'', T'. By induction hypothesis (1), $E \vdash Q' : {}^{Z'}T'$. By (Proc Amb), $E \vdash M[Q'] : {}^{Z}T$. Part (2) is symmetric.

(Struct Action) Then $P = M.P'$, $Q = M.Q'$, and $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Action), with $E \vdash M : Cap[{}^{Z}T]$, $E \vdash P' : {}^{Z}T$. By induction hypothesis (1), $E \vdash Q' : {}^{Z}T$. By (Proc Action), $E \vdash M.Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Input) In this case, $P = (n_1{:}W_1, \ldots, n_k{:}W_k).P'$, $P' \equiv Q'$, and $Q = (n_1{:}W_1, \ldots, n_k{:}W_k).Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Input), with $E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P' : {}^{Z}T$, where $T = W_1 \times \cdots \times W_k$. By induction hypothesis, we have $E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash Q' : {}^{Z}T$. By (Proc Input), we have $E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).Q' : {}^{Z}T$. Part (2) is symmetric.

(Struct Go) Then $P = go\ N.M[P']$, $Q = go\ N.M[Q']$, and $P' \equiv Q'$. For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Go),

with $E \vdash N : Cap[^{Z''}S]$, $E \vdash M : Amb^{Y'\,Z''}[^{Z'}T']$ and $E \vdash P' : {}^{Z'}T'$, for some S, Y', Z', Z'', and T'. By induction hypothesis (1), $E \vdash Q' : {}^{Z'}T'$. By (Proc Go), $E \vdash go\ N.M[Q'] : {}^{Z}T$. Part (2) is symmetric.

(Struct Par Comm) Then $P = P' \mid P''$ and $Q = P'' \mid P'$.

For (1), assume $E \vdash P' \mid P'' : {}^{Z}T$. This must have been derived from $E \vdash P' : {}^{Z}T$ and $E \vdash P'' : {}^{Z}T$. By (Proc Par), $E \vdash P'' \mid P' : {}^{Z}T$. Hence, $E \vdash Q : {}^{Z}T$.

Part (2) is symmetric.

(Struct Par Assoc) Then $P = (P' \mid P'') \mid P'''$ and $Q = P' \mid (P'' \mid P''')$.

For (1), assume $E \vdash (P' \mid P'') \mid P''' : {}^{Z}T$. This must have been derived from (Proc Par) twice, with $E \vdash P' : {}^{Z}T$, $E \vdash P'' : {}^{Z}T$, and $E \vdash P''' : {}^{Z}T$. By (Proc Par) twice, $E \vdash P' \mid (P'' \mid P''') : {}^{Z}T$. Hence $E \vdash Q : {}^{Z}T$.

Part (2) is symmetric.

(Struct Repl Par) Then $P = !P'$ and $Q = P' \mid !P'$.

For (1), assume $E \vdash !P' : {}^{Z}T$. This must have been derived from (Proc Repl), with $E \vdash P' : {}^{Z}T$. By (Proc Par), $E \vdash P' \mid !P' : {}^{Z}T$. Hence, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash P' \mid !P' : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash P' : {}^{Z}T$ and $E \vdash !P' : {}^{Z}T$. Hence, $E \vdash P : {}^{Z}T$.

(Struct Res Res) In this case, we have $P = (\nu n_1{:}W_1)(\nu n_2{:}W_2)P'$ and $Q = (\nu n_2{:}W_2)(\nu n_1{:}W_1)P'$ with $n_1 \neq n_2$.

For (1), assume $E \vdash (\nu n_1{:}W_1)(\nu n_2{:}W_2)P' : {}^{Z}T$. This must have been derived from (Proc Res), with $E, n_1{:}Amb^{Y_1\,Z_1'}[^{Z_1}T_1], n_2{:}Amb^{Y_2\,Z_2'}[^{Z_2}T_2] \vdash P' : {}^{Z}T$, where $W_1 = Amb^{Y_1\,Z_1'}[^{Z_1}T_1]$ and $W_2 = Amb^{Y_2\,Z_2'}[^{Z_2}T_2]$. By Lemma 14, we have $E, n_2{:}Amb^{Y_2\,Z_2'}[^{Z_2}T_2], n_1{:}Amb^{Y_1\,Z_1'}[^{Z_1}T_1] \vdash P' : {}^{Z}T$. By (Proc Res) twice we have $E \vdash (\nu n_2{:}W_2)(\nu n_1{:}W_1)P' : {}^{Z}T$. Part (2) is symmetric.

(Struct Res Par) Then $P = (\nu n{:}W)(P' \mid P'')$ and $Q = P' \mid (\nu n{:}W)P''$, with $n \notin fn(P')$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res), with $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P' \mid P'' : {}^{Z}T$ and $W =$

$Amb^{Y'\,Z''}[^{Z'}T']$, and from (Proc Par), with $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P' : {}^{Z}T$ and $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P'' : {}^{Z}T$. By Lemma 15, since $n \notin fn(P')$, we have $E \vdash P' : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}Amb^{Y'\,Z''}[^{Z'}T'])P'' : {}^{Z}T$. By (Proc Par) we have $E \vdash P' \mid (\nu n{:}Amb^{Y'\,Z''}[^{Z'}T'])P'' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash Q : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash P' : {}^{Z}T$ and $E \vdash (\nu n{:}W)P'' : {}^{Z}T$, and from (Proc Res), with $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P'' : {}^{Z}T$ and $W = Amb^{Y'\,Z''}[^{Z'}T']$. By Lemma 12, $n \notin dom(E)$. By Lemma 16, $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P' : {}^{Z}T$. By (Proc Par), $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P' \mid P'' : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}Amb^{Y'\,Z''}[^{Z'}T'])(P' \mid P'') : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Res Amb) Then $P = (\nu n{:}W)m[P']$ and $Q = m[(\nu n{:}W)P']$, with $n \neq m$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res) with $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash m[P'] : {}^{Z}T$ with $W = Amb^{Y'\,Z''}[^{Z'}T']$, and from (Proc Amb) with $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash m : Amb^{Y_m\,Z'_m}[^{Z_m}T_m]$ and $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P' : {}^{Z_m}T_m$ for some Y_m, Z_m, Z'_m, T_m. By (Proc Res) we have $E \vdash (\nu n{:}Amb^{Y'\,Z''}[^{Z'}T'])P' : {}^{Z_m}T_m$. By Lemma 15, $n \neq m$ implies $E \vdash m : Amb^{Y_m\,Z'_m}[^{Z_m}T_m]$. By (Proc Amb), $E \vdash m[(\nu n{:}Amb^{Y'\,Z''}[^{Z'}T'])P'] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash Q : {}^{Z}T$. This must have been derived from (Proc Amb) with $E \vdash m : Amb^{Y_m\,Z'_m}[^{Z_m}T_m]$ and $E \vdash (\nu n{:}W)P' : {}^{Z_m}T_m$, and from (Proc Res), with $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash P' : {}^{Z_m}T_m$ and $W = Amb^{Y'\,Z''}[^{Z'}T']$. By Lemma 12, $n \notin dom(E)$. By Lemma 16, $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash m : Amb^{Y_m\,Z'_m}[^{Z_m}T_m]$. By (Proc Amb), we can derive $E, n{:}Amb^{Y'\,Z''}[^{Z'}T'] \vdash m[P'] : {}^{Z}T$. By (Proc Res), we can derive $E \vdash (\nu n{:}Amb^{Y'\,Z''}[^{Z'}T'])m[P'] : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Zero Par) Then $P = P' \mid \mathbf{0}$ and $Q = P'$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Par) with $E \vdash P' : {}^{Z}T$ and $E \vdash \mathbf{0} : {}^{Z}T$. Hence, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash P' : {}^{Z}T$. By Lemma 11, $E \vdash \diamond$. By (Proc Zero), $E \vdash \mathbf{0} : {}^{Z}T$. By (Proc Par), $E \vdash P' \mid \mathbf{0} : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Zero Res) Then $P = (\nu n{:}Amb^{Y'\,Z''}[^{Z'}T'])\mathbf{0}$ and $Q = \mathbf{0}$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Res) with $E, n{:}Amb^{Y'\,Z''}[{}^{Z'}T'] \vdash \mathbf{0} : {}^{Z}T$. By Lemma 15, $E \vdash \mathbf{0} : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash \mathbf{0} : {}^{Z}T$. We may assume that the bound name n does not occur in $dom(E)$. By Lemma 16, $E, n{:}Amb^{Y'\,Z''}[{}^{Z'}T'] \vdash \mathbf{0} : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}Amb^{Y'\,Z''}[{}^{Z'}T'])\mathbf{0} : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Zero Repl) Then $P = !\mathbf{0}$ and $Q = \mathbf{0}$.

For (1), assume $E \vdash P : {}^{Z}T$. By Lemma 11, $E \vdash \diamond$. By (Proc Zero), $E \vdash \mathbf{0} : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash \mathbf{0} : {}^{Z}T$. By (Proc Repl), $E \vdash !\mathbf{0} : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct ϵ) Then $P = \epsilon.P'$ and $Q = P'$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Action) with $E \vdash \epsilon : Cap[{}^{Z}T]$ and $E \vdash P' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash P' : {}^{Z}T$. By Lemma 11, $E \vdash \diamond$. By (Exp ϵ), $E \vdash \epsilon : Cap[{}^{Z}T]$. By (Proc Action), $E \vdash \epsilon.P' : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct .) Then $P = (M.M').P'$ and $Q = M.M'.P'$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Action) with $E \vdash P' : {}^{Z}T$, $E \vdash M.M' : Cap[{}^{Z}T]$. The latter must have come from (Exp .) with $E \vdash M : Cap[{}^{Z}T]$ and $E \vdash M' : Cap[{}^{Z}T]$, By (Proc Action) twice, $E \vdash M.(M'.P') : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash Q : {}^{Z}T$. This must have been derived from (Proc Action), twice, with $E \vdash M : Cap[{}^{Z}T]$, $E \vdash M' : Cap[{}^{Z}T]$, and $E \vdash P' : {}^{Z}T$. By (Exp .), $E \vdash M.M' : Cap[{}^{Z}T]$. By (Proc Action), $E \vdash (M.M').P' : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$.

(Struct Go ϵ) Then $P = go\ \epsilon.M[P']$ and $Q = M[P']$.

For (1), assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Go), with $E \vdash \epsilon : Cap[{}^{Z''}S]$, $E \vdash M : Amb^{Y'\,Z''}[{}^{Z'}T']$, and $E \vdash P' : {}^{Z'}T'$. By (Proc Amb), $E \vdash M[P'] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

For (2), assume $E \vdash Q : {}^{Z}T$. This must have been derived using (Proc Amb), with $E \vdash M : Amb^{Y'\,Z''}[{}^{Z'}T']$ and $E \vdash P' : {}^{Z'}T'$. By Lemma 11, $E \vdash \diamond$. By (Exp ϵ), $E \vdash \epsilon : Cap[{}^{Z''}S]$, for some S. By (Proc Go), $E \vdash go\ \epsilon.n[P'] : {}^{Z}T$, that is, $E \vdash P : {}^{Z}T$. □

Proof of Theorem 2 *If $E \vdash P : {}^{Z}T$ and $P \to Q$ then $E \vdash Q : {}^{Z}T$.*

Proof By induction on the derivation of $P \to Q$.

(Red In) Then $P = n[in\ m.P' \mid P''] \mid m[P'''']$ and $Q = m[n[P' \mid P''] \mid P'''']$. Assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Par), with $E \vdash n[in\ m.P' \mid P''] : {}^{Z}T$ and $E \vdash m[P''''] : {}^{Z}T$. The former must have been derived from (Proc Amb), with $E \vdash n : Amb^{Y_n\, Z'_n}[{}^{Z_n}T_n]$ and $E \vdash in\ m.P' \mid P'' : {}^{Z_n}T_n$, for some Y_n, Z_n, Z'_n, and T_n, while the latter must have been derived from (Proc Amb) with $E \vdash m : Amb^{Y_m\, Z'_m}[{}^{Z_m}T_m]$ and $E \vdash P'''' : {}^{Z_m}T_m$, for some Y_m, Z_m, Z'_m, and T_m. Moreover, $E \vdash in\ m.P' \mid P'' : {}^{Z_n}T_n$ must come from (Proc Par) with $E \vdash in\ m.P' : {}^{Z_n}T_n$ and $E \vdash P'' : {}^{Z_n}T_n$. Finally, $E \vdash in\ m.P' : {}^{Z_n}T_n$ must come from $E \vdash in\ m : Cap[{}^{Z_n}T_n]$ and $E \vdash P' : {}^{Z_n}T_n$. (Hence $Z_n = \curvearrowright$, since the former judgment can only be derived using (Exp In).) By (Proc Par), we have $E \vdash P' \mid P'' : {}^{Z_n}T_n$, and by (Proc Amb) we can derive $E \vdash n[P' \mid P''] : {}^{Z_m}T_m$. Then, by (Proc Par), we have $E \vdash n[P' \mid P''] \mid P'''' : {}^{Z_m}T_m$. By (Proc Amb) we can derive $E \vdash m[n[P' \mid P''] \mid P''''] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Out) Then $P = m[n[out\ m.P' \mid P''] \mid P'''']$ and $Q = n[P' \mid P''] \mid m[P'''']$. Assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Amb) using $E \vdash m : Amb^{Y_m\, Z'_m}[{}^{Z_m}T_m]$ and $E \vdash n[out\ m.P' \mid P''] \mid P'''' : {}^{Z_m}T_m$ for some Y_m, Z_m, Z'_m, and T_m, and from (Proc Par) using $E \vdash n[out\ m.P' \mid P''] : {}^{Z_m}T_m$ and $E \vdash P'''' : {}^{Z_m}T_m$. The former must have been derived using (Proc Amb) from $E \vdash n : Amb^{Y_n\, Z'_n}[{}^{Z_n}T_n]$ and $E \vdash out\ m.P' \mid P'' : {}^{Z_n}T_n$ for some Y_n, Z_n, Z'_n, and T_n, and using (Proc Par) from $E \vdash out\ m.P' : {}^{Z_n}T_n$ and $E \vdash P'' : {}^{Z_n}T_n$. The former must have been derived using (Proc Action) from $E \vdash out\ m : Cap[{}^{Z_n}T_n]$ and $E \vdash P' : {}^{Z_n}T_n$ (Hence $Z_n = \curvearrowright$, since the former judgment can only be derived using (Exp Out).) By (Proc Par), $E \vdash P' \mid P'' : {}^{Z_n}T_n$. By (Proc Amb), $E \vdash n[P' \mid P''] : {}^{Z}T$. By (Proc Amb), $E \vdash m[P''''] : {}^{Z_n}T_n$. By (Proc Par), $E \vdash n[P' \mid P''] \mid m[P''''] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Open) Then $P = open\ n.P' \mid n[P'']$ and $Q = P' \mid P''$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Par) from $E \vdash open\ n.P' : {}^{Z}T$ and $E \vdash n[P''] : {}^{Z}T$. The former must have been derived using (Proc Action) with $E \vdash open\ n : Cap[{}^{Z}T]$ and $E \vdash P' : {}^{Z}T$, while the latter must have been derived using (Proc

Amb) with $E \vdash n : Amb^{Y'\,Z''}[^{Z'}T']$ and $E \vdash P'' : {}^{Z'}T'$ for some Y', Z', Z'', and T'. The judgment $E \vdash open\, n : Cap[^{Z}T]$ must have been derived using (Exp Open) from $E \vdash n : Amb^{\circ\,Z'''}[^{Z}T]$ By Lemma 13, $Amb^{Y'\,Z''}[^{Z'}T'] = Amb^{\circ\,Z'''}[^{Z}T]$, and so $Y' = \circ$, $Z'' = Z'''$, $Z' = Z$, and $T' = T$. Hence, by (Proc Par), $E \vdash P' \mid P'' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Comm) Then $P = (n_1{:}W_1, \ldots, n_k{:}W_k).P' \mid \langle M_1, \ldots, M_k\rangle$ and $Q = P'\{n_1{\leftarrow}M_1, \ldots, n_k{\leftarrow}M_k\}$. Assume $E \vdash P : {}^{Z}T$. This must have been derived from (Proc Par) with $E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P' : {}^{Z}T$ and $E \vdash \langle M_1, \ldots, M_k\rangle : {}^{Z}T$. The former must have been derived from (Proc Input) with $E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P' : {}^{Z}T$ and $T = W_1 \times \cdots W_k$. The latter judgment $E \vdash \langle M_1, \ldots, M_k\rangle : {}^{Z}T$ must have been derived from (Proc Output) with $E \vdash M_i : W_i'$ for each $i \in 1..k$, and $T = W_1' \times \cdots \times W_k'$. Hence $W_i' = W_i$ for each $i \in 1..k$. By k applications of Lemma 17, we get $E \vdash P'\{n_1{\leftarrow}M_1, \ldots, n_k{\leftarrow}M_k\} : {}^{Z}T$.

(Red Go In) Here $P = go\,(in\, m.N).n[P_n] \mid m[P_m]$ and $Q = m[go\, N.n[P_n] \mid P_m]$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Par) from $E \vdash go\,(in\, m.N).n[P_n] : {}^{Z}T$ and $E \vdash m[P_m] : {}^{Z}T$. The former must have been derived using (Proc Go) with $E \vdash in\, m.N : Cap[^{\frown}S]$, $E \vdash n : Amb^{Y_n\,\frown}[^{Z_n}T_n]$, and $E \vdash P_n : {}^{Z_n}T_n$, for some S, Y_n, Z_n, and T_n, and the latter must have been derived using (Proc Amb) with $E \vdash m : Amb^{Y_m\,Z_m'}[^{Z_m}T_m]$ and $E \vdash P_m : {}^{Z_m}T_m$ for some Y_m, Z_m, Z_m', and T_m. Moreover, the judgment $E \vdash in\, m.N : Cap[^{\frown}S]$ must have been derived using (Exp .) from $E \vdash in\, m : Cap[^{\frown}S]$ and $E \vdash N : Cap[^{\frown}S]$. By (Proc Go) and (Proc Par), $E \vdash go\, N.n[P_n] \mid P_m : {}^{Z_m,T_m}$. By (Proc Amb), we get $E \vdash m[go\, N.n[P_n] \mid P_m] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Go Out) Here $P = m[go\,(out\, m.N).n[P_n] \mid P_m]$ and $Q = go\, N.n[P_n] \mid m[P_m]$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Amb) from $E \vdash m : Amb^{Y_m\,Z_m'}[^{Z_m}T_m]$ and $E \vdash go\ (out\ m.N).n[P_n] \mid P_m : {}^{Z_m}T_m$ for some Y_m, Z_m, Z_m', and T_m, and from (Proc Par) with $E \vdash go\ (out\ m.N).n[P_n] : {}^{Z_m}T_m$ and $E \vdash P_m : {}^{Z_m}T_m$. The former must have been derived using (Proc Go) from $E \vdash out\, m.N : Cap[^{\frown}S]$, $E \vdash n : Amb^{Y_n\,\frown}[^{Z_n}T_n]$, and $E \vdash P_n : {}^{Z_n}T_n$ for some S, Y_n, Z_n, and T_n. The judgment $E \vdash out\, m.N : Cap[^{\frown}S]$ must have been derived using (Proc .) using $E \vdash out\, m : Cap[^{\frown}S]$ and $E \vdash N : Cap[^{\frown}S]$. By (Proc Go), $E \vdash go\, N.n[P_n] : {}^{Z}T$. By (Proc Amb), $E \vdash m[P_m] : {}^{Z}T$. By (Proc Par), $E \vdash go\, N.n[P_n] \mid m[P_m] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Res) Here $P = (\nu n{:}W)P'$ and $Q = (\nu n{:}W)Q'$ with $P' \to Q'$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Res) from $E, n{:}Amb^{Y'\,Z''}[{}^{Z'}T'] \vdash P' : {}^{Z}T$ with $W = Amb^{Y'\,Z''}[{}^{Z'}T']$. By induction hypothesis, $E, n{:}Amb^{Y'\,Z''}[{}^{Z'}T'] \vdash Q' : {}^{Z}T$. By (Proc Res), $E \vdash (\nu n{:}Amb^{Y'\,Z''}[{}^{Z'}T'])Q' : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Amb) Here $P = n[P']$ and $Q = n[Q']$ with $P' \to Q'$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Amb) from $E \vdash n : Amb^{Y'\,Z''}[{}^{Z'}T']$ and $E \vdash P' : {}^{Z'}T'$. By induction hypothesis, $E \vdash Q' : {}^{Z'}T'$. By (Proc Amb), $E \vdash n[Q'] : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red Par) Here $P = P' \mid R$ and $Q = Q' \mid R$ with $P' \to Q'$. Assume $E \vdash P : {}^{Z}T$. This must have been derived using (Proc Par) from $E \vdash P' : {}^{Z}T$ and $E \vdash R : {}^{Z}T$. By induction hypothesis, $E \vdash Q' : {}^{Z}T$. By (Proc Par), $E \vdash Q' \mid R : {}^{Z}T$, that is, $E \vdash Q : {}^{Z}T$.

(Red $\equiv$) Here $P \equiv P'$, $P' \to Q'$, and $Q' \equiv Q$. Assume $E \vdash P : {}^{Z}T$. By Proposition 18, $E \vdash P' : {}^{Z}T$. By induction hypothesis, $E \vdash Q' : {}^{Z}T$. By Proposition 18, $E \vdash Q : {}^{Z}T$. □