Mobility and Security

Luca Cardelli

Microsoft Research

Lecture Notes for the Marktoberdorf Summer School 1999,
from works by Luca Cardelli and Andrew D. Gordon

Abstract. We discuss the computational aspects of wide area networks, and we describe various facets of a process calculus devised to embody mobility, security, and wide area network semantics. These lecture notes are an abridged version of [8, 11, 27, 12, 13].

Part I: Wide Area Computation

1 Introduction

The Internet and the World-Wide-Web provide a computational infrastructure that spans the planet. It is appealing to imagine writing programs that exploit this global infrastructure. Unfortunately, the Web violates many familiar assumptions about the behavior of distributed systems, and demands novel and specialized programming techniques. In particular, three phenomena that remain largely hidden in local area network architectures become readily observable on the Web:

- ***(A) Virtual locations***. Because of the presence of potential attackers, barriers are erected between mutually distrustful administrative domains. Therefore, a program must be aware of where it is, and of how to move or communicate between different domains. The existence of separate administrative domains induces a notion of virtual locations and of virtual distance between locations.
- ***(B) Physical locations***. On a planet-size structure, the speed of light becomes tangible. For example, a procedure call to the antipodes requires at least 1/10 of a second, independently of future improvements in networking technology. This absolute lower bound to latency induces a notion of physical locations and physical distance between locations.
- ***(C) Bandwidth fluctuations***. A global network is susceptible to unpredictable congestion and partitioning, which result in fluctuations or temporary interruptions of bandwidth. Moreover, mobile devices may perceive bandwidth changes as a consequence of physical movement. Programs need to be able to observe and react to these fluctuations.

These features may interact among themselves. For example, bandwidth fluctuations may be related to physical location because of different patterns of day and night network utilization, and to virtual location because of authentication and encryption across domain boundaries. Virtual and physical locations are often related, but need not coincide.

In addition, another phenomenon becomes unobservable on the Web:

- ***(D) Failures***. On the Web, there is no practical upper bound to communication delays. In particular, failures become indistinguishable from long delays, and thus undetectable. Failure recovery becomes indistinguishable from intermittent connectivity. Furthermore, delays (and, implicitly, failures) are frequent and unpredictable.

These four phenomena determine the set of *observables* of the Web: the events or states that can be in principle detected. Observables, in turn, influence the basic building blocks of computation. In moving from local area networks to wide area networks, the set of observables changes, and so does the computational model, the programming constructs, and the kind of programs one can write. The question of how to "program the Web" reduces to the question of how to program with the new set of observables provided by the Web.

At least one general technique has emerged to cope with the observables characteristic of a wide area network such as the Web. *Mobile computation* is the notion that running programs need not be forever tied to a single network node. Mobile computation can deal in original ways with the phenomena described above:

- ***(A) Virtual locations***. Given adequate trust mechanisms, mobile computations can cross barriers and move between virtual locations. Barriers are designed to impede access, but when code is allowed to cross them, it can access local resources without the impediment of the barriers.
- ***(B) Physical locations***. Mobile computations can move between physical locations, turning remote calls into local calls, and thus avoiding the latency limit.
- ***(C) Bandwidth fluctuations***. Mobile computations can react to bandwidth fluctuations, either by moving to a better-placed location, or by transferring code that establishes a customized protocol over a connection.
- ***(D) Failures***. Mobile computations can run away from anticipated failures, and can move around presumed failures.

Mobile computation is also strongly related to recent hardware advances, since computations move implicitly when carried on portable devices. In this sense, we cannot avoid the issues raised by mobile computation: more than an avant-garde software technique, it is an existing hardware reality.

2 Three Mental Images

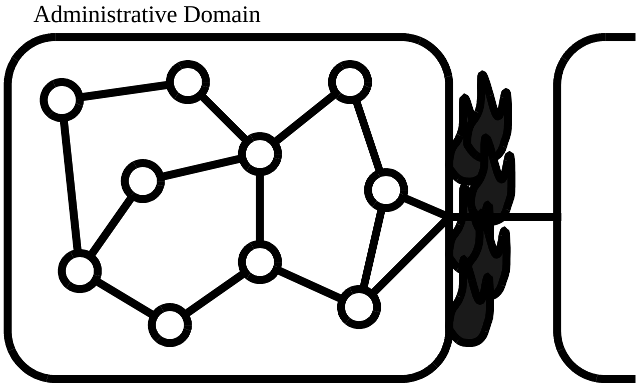
We begin by comparing and contrasting three *mental images*; that is, three abstracted views of distributed computation. From the differences between these mental images we derive the need for new approaches to global computation.

2.1 Local Area Networks

The first mental image corresponds to the now standard, and quickly becoming obsolete, model of computation over local area networks.

When workstations and PCs started replacing mainframes, local networks were invented to connect autonomous computers for the purpose of resource sharing. A typical local area network consists of a collection of computers of about the same power (within a couple of hardware generations) and of network links of about the same bandwidth and latency. This environment is not always completely uniform: specialized machines may operate as servers or as engineering workstations, and specialized subnetworks may offer special services. Still, by and large, the structure of a LAN can be depicted as the uniform network of nodes (computers) and links (connections) in Mental Image 1.

A main property of such a network is its predictability. Communication delays are bounded, and processor response time can be estimated. Therefore, link and process failures can be detected by time-outs and by "pinging" nodes.



Mental Image 1: Local Area Network

Another important property of local area networks is that they are usually well-administered and, in recent times, protected against attack. Network administrators have the task of keeping the network running and protect it against infiltration. In the picture, the boundary line represents an *administrative domain*, and the flames represent the protection provided by a *firewall*. Protection is necessary because local area networks are rarely completely disconnected: they usually have slower links to the outside world, which are however enough to make administrators nervous about infiltration.

The architecture of local area networks is very different from the older, highly centralized, mainframe architecture. This difference, and the difficulties implied by it, resulted in the emergence of novel distributed computing techniques, such as remote-procedure-call, client-server architecture, and distributed object-oriented programming. The combined aim and effect of these techniques is to make the programming and application environment stable and uniform (as in mainframes). In particular, the network topology is carefully hidden so that any two computers can be considered as lying one logical step apart. Moreover, computers can be considered immobile; for example, they usually preserve their network address when physically moved.

Even in this relatively static environment, the notion of mobility has gradually acquired prominence, in a variety of forms. Control mobility, found in RPC (Remote Procedure Call) and RMI (Remote Method Invocation) mechanisms, is the notion that a thread of control moves (in principle) from one machine to another and back. Data mobility is achieved in RPC/RMI by linearizing, transporting, and reconstructing data across machines. Link mobility is the ability to transmit the end-points of network channels, or remote object proxies. Object mobility is the ability to move objects between different servers, for example for load balancing purposes. Finally, in Remote Execution, a computations can be shipped for execution to a server (this is an early version of code mobility, proposed as an extension of RPC [41]).

In recent years, distributed computing has been endowed with greater mobility properties and easier network programming. Techniques such as Object Request Brokers have emerged to abstract over the location of objects providing certain services. Code mobility has emerged in Tcl and other scripting languages to control network applications. Agent mobility has been pioneered in Telescript [43], aimed towards a uniform (although wide area) network of services. Closure mobility (the mobility of active and connected entities) has been investigated in Obliq [7].

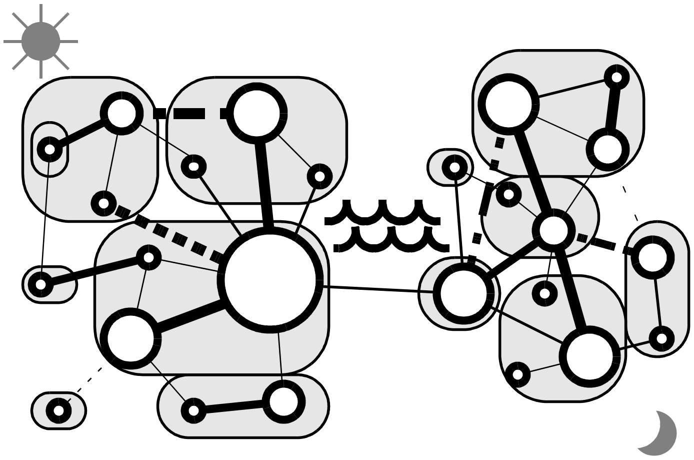
In due time, local area network techniques would have smoothly and gradually evolved towards deployment on wide area networks, e.g. as was explicitly attempted by the CORBA effort. But, suddenly, a particular wide area network came along that radically changed the fundamental assumptions of distributed computing and its pace of progress: the Web.

2.2 *Wide Area Networks*

Global computing evolved over the span of a few decades in the form of the Internet. But it was not until the emergence of the Web that the peculiar characteristics of the Internet were exposed in a way that anybody could verify with just a few mouse clicks. For clarity and simplicity we will refer to the Web as the primary global information infrastructure, although it was certainly not the first one.

We should remember that the notions of a global address space and of a global file system have been popular at times as extensions of the mainframe architecture to wide area networks. The first obvious feature of the Web is that, although it forms a global computational resource, it is nothing like a global mainframe, nor an extension of it. The Web does not support a global (updatable) file system and, although it supports a global addressing mechanism, it does not guarantee the integrity of addressing. The Web has no single reliable component, but it also has no single failure point; it is definitely not the centralized all-powerful mainframe of 1950's science fiction novels that could be shut off by attacking its single "brain".

The fact that the Web is not a mainframe is not a big concern; we have already successfully tackled distributed computing based on LANs. More distressing is the fact that the Web does not behave like a LAN either. Many proposals have emerged along the lines of extending LAN concepts to a global environment; that is, in turning the Internet into a distributed address space, or a distributed file system. However, since the global environment does not have the stability properties of a LAN, this can be achieved only by introducing redundancy (for reliability), replication (for quality of service), and scalability (for management) at many different levels. Things might have evolved in this direction, but this is not the way the Web came to be. The Web is, almost by definition, unreliable, unpredictable, and unmanageable as a whole, and was not designed with LAN-like guarantees of service.



Mental Image 2: Wide Area Network (for example, the Web)

Therefore, the main problem with the Web is that it is not just a big LAN, otherwise, modulo issues of scale, we would already know how to deal with it. There are several ways in which the Web is not a big LAN, and we will describe them shortly. But the fundamental reason is that, unlike a LAN, the Web is not centrally administered. Instead, it is a dynamic collection of countless independent administrative domains, all widely different and mutually distrustful. This is represented in Mental Image 2.

In that picture, computers differ greatly in power and availability, while network links differ greatly in capacity and reliability. Large physical distances have visible effects, and so do time zones. The architecture of a wide area network is yet again fundamentally different from that of a local area network. Most prominently, the network topology is dynamic and nontrivial. Computers become intrinsically mobile: they require different addressing when physically moved across administrative boundaries. Techniques based on mobility become more important and sometimes essential. For example, mobile Java applets provided the first disciplined mechanism for running code able to (and allowed to) systematically penetrate other people's firewalls. Countless projects have emerged in the last few years with the aim of supporting mobile computation over wide areas, and are beginning to be consolidated.

At this point, our architectural goal might be to devise techniques for managing computation over an unreliable collection of far-flung computers. However, this is not yet the full picture. Not only are network links and nodes widely dispersed and unreliable; they are not even liable to stay put, as we discuss next.

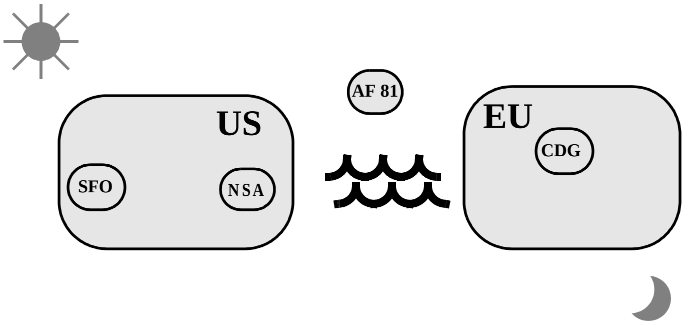
2.3 Mobile Computing

A different global computing paradigm has been evolving independently of the Web. Instead of connecting together all the LANs in the world, another way of extending the reach of a LAN is to move individual computers and other gadgets from one LAN to another, dynamically.

We discussed in the Introduction how the main characteristics of the Web point towards mobile comput*ation*. However, that is meant as mobile computation over a fixed (although possibly flaky) network. A more interesting picture emerges when the very components of the network can move about. This is the field of mobile comput*ing*. Today, laptops and personal organizers routinely move about; in the future entire networks will go mobile (as in IBM's Personal Area Network). Existing examples of this kind of mobility include: a smart card entering a network computer slot; an active badge entering a room; a wireless PDA or laptop entering a building; a mobile phone entering a phone cell.

We could draw a picture similar to Mental Image 1, but with mobile devices moving within the confines of a single LAN. This notion of a dynamic LAN is a fairly minor extension of the basic LAN concepts, and presents few conceptual problems (wireless LANs are already common). A much more interesting picture emerges when we think of mobile gadgets over a WAN, because administrative boundaries and multiple access pathways then interact in complex ways, as anybody who travels with a laptop knows all too well.

Mental Image 3 focuses on two domains: the United States and the European Union, each enclosed by a political boundary that regulates the movement of people and computers. Within a political boundary, private companies and public agencies may further regulate the flow of people and devices across their doors. Over the Atlantic we see a third domain, representing Air France flight 81 travelling from San Francisco to Paris. AF81 is a very active mobile computational environment: it is full of people working with their laptops and possibly connecting to the Internet through airphones. (Not to mention the hundreds of computers that control the airplane and let it communicate with a varying stream of ground stations.)



Mental Image 3: Mobile Computing

Abstracting a bit from people and computation devices, we see here a hierarchy of boundaries that enforce controls and require permissions for crossing. Passports are required to cross political boundaries, tickets are required for airplanes, and special clearances are required to enter (and exit!) agencies such as the NSA. Sometimes, whole mobile boundaries cross in and out of other boundaries and similarly need permissions, as the mobile environment of AF81 needs permission to enter an airspace. On the other hand, once an entity has been allowed across a boundary, it is fairly free to roam within the confines of the boundary, until another boundary needs to be crossed.

2.4 General Mobility

We have described two different notions of mobility. The first, *mobile computation*, has to do with virtual mobility (mobile software). The second, *mobile computing*, has to do with physical mobility (mobile hardware). These two fields are today almost disconnected, the first dominated by a software community, and the second dominated by a hardware community. However, the borders between virtual and physical mobility are fuzzy, and eventually we will have to treat all kinds of mobility in a uniform way. Here are two examples where the different forms of mobility interact.

The first example is one of virtual mobility achieved by physical means. Consider a software agent in a laptop. The agent can move by propagating over the network, but can also move by being physically transported with the laptop from one location to another. In the first case, the agent may have to undergo security checks (e.g., bytecode verification) when it crosses administrative domains. In the second case the agent may have to undergo security checks (e.g., virus detection) when the laptop is physically allowed inside a new administrative domain. Do we need two completely separate security infrastructures for these two cases, or can we somehow find a common principle? A plausible security policy for a given domain would be that a physical barrier (a building door) should provide the same security guarantees as a virtual barrier (a firewall).

The second example is one of physical mobility achieved by virtual means. Software exists that allows remote control of a computer, by bringing the screen of a remote computer on a local screen. The providers of such software may claim that this is just as good as moving the computer physically, e.g. to access its local data. Moreover, if the remote computer has a network connection, this is also equivalent to “stringing wire” from the remote location, since the remote network is now locally accessible. For example, using remote control over a phone line to connect from home to work where a high-bandwidth Internet connection is available, is almost as good as having a high-bandwidth Internet connection brought into the home.

The other side of the coin of being mobile is of becoming disconnected or intermittently connected. Even barring flaky networks, intermittent connectivity can be caused by physical movement, for example when a wireless user moves into some form of Faraday cage. More interestingly, intermittent connectivity may be caused by virtual movement, for example when an agent moves in and out of an administrative domain that does not allow communication. Neither case is really a failure of the infrastructure; in both cases, lack of connectivity may in fact be a desirable security feature. Therefore, we have to assume that intermittent connectivity, caused equivalently by physical or virtual means, is an essential feature of mobility.

In the future we should be prepared to see increased interactions between virtual and physical mobility, and we should develop frameworks where we can discuss and manipulate these interactions.

2.5 Barriers and Action-at-a-Distance

The unifying difficulty in both mobile computing and mobile computation is the proliferation of barriers, and the problems involved in crossing them. This central difficulty implies that we must regard barriers as fundamental features of our computational models.

In general, we can forget about the notion of *action-at-a-distance computing*: the idea that resources are available transparently at any time, no matter how far away. Instead, we have to get used to the notion that movement and communication are step-by-step activities, and that they are visibly so: the multiple steps involved cannot be hidden, collapsed, or rendered atomic.

The action-at-a-distance paradigm is still prevalent within LANs, and this is another reason why LANs are different from WANs, where such an assumption cannot hold.

2.6 WAN Postulates

We summarize this section by a collection of postulates that capture the main properties of the reality we are interested in modeling:

- Separate locations exist.
- Different locations have different properties, hence both people and programs will want to move between them.
- Barriers to mobility will be erected to preserve the properties of certain locations.
- Some people and some programs will still need to cross those barriers.

The point of these postulates is to stress that mobility and barrier crossing are inevitable requirements of our current and future computing infrastructure.

The observables that are characteristic of wide area networks have the following implications:

- Distinct virtual locations are observed because of the existence of distinct administrative domains, which are produced by the inevitable existence of attackers. Distinct virtual locations preclude the unfettered execution of actions across domains, and require a security model.
- Distinct physical locations are observed, over large distances, because of the inevitable latency limit given by the speed of light. Distinct physical locations preclude instantaneous action at a distance, and require a mobility model.
- Bandwidth fluctuations (including hidden failures) are observed because of the inevitable exercise of free will by network users, both in terms of communication and movement. Bandwidth fluctuations preclude reliance on response time, and require an asynchronous communication model.

3 Modeling Wide Area Computation

Section 2 was dedicated to showing that the reality of mobile computation over a WAN does not fall into familiar categories. Therefore, we need to invent a new model that can help us in understanding and eventually in taking advantage of this reality.

3.1 Barriers

We believe that the most fundamental new notion is that of barriers; this is the most prominent aspect of post-LAN computing environments.

Many of the basic features of WANs have to do with barriers: ***Locality*** (the existence of different virtual or physical locations, and the notion of being in the same or different locations) is induced by a topology of barriers. ***Mobility*** is barrier crossing. ***Security*** has to do with the ability or inability to cross barriers. ***Communication*** is partitioned by barriers: local communication happens within barriers, while long-distance communication is a combination of local communication and movement across barriers. ***Action at a distance*** (immediate interaction across many barriers) is forbidden.

We have chose barriers as the most important feature of an abstract model of computation for wide area networks, the Ambient Calculus [11], which we briefly outline here and investigate in the remaining sections.

3.2 Ambients

The current literature on wide area network languages can be broadly classified into agent-based languages (e.g., Telescript [43]), and place-based languages (e.g., Linda [14]). An ambient is a generalization of both notions. Like an agent, an ambient can move across places (also represented by ambients) where it can interact with other agents. Like a place, an ambient supports local undirected communication, and can receive messages (also represented by ambients) from other places. Ambients can be arbitrarily nested, generalizing the limited place-agent-data nesting of most agent languages, and the nesting of places allowed in some Linda dialects.

Briefly, an *ambient* is a place that is delimited by a boundary and where multi-threaded computation happens. Each ambient has a *name*, a collection of local *processes*, and a collection of *subambients*. Ambients can move in and out of other ambients, subject to *capabilities* that are associated with ambient names. Ambient names are unforgeable, this fact being the most basic security property.

In further detail, an ambient has the following main characteristics.

- An ambient is a *bounded* place where computation happens.

  If we want to move computations easily we must be able to determine what parts should move. A boundary determines what is inside and what is outside an ambient, and therefore determines what moves. A boundary implies some flexible addressing scheme that can denote entities across the boundary; examples are symbolic links, URLs (Uniform Resource Locators) and Remote Procedure Call proxies. Flexible addressing is what enables, or at least facilitates, mobility. It is also, of course, a cause of problems when the addressing links are “broken”.

- Ambients can be nested within other ambients, forming a tree structure.

  As we discussed, administrative domains are (often) organized hierarchically. Mobility is represented as navigation across a hierarchy of ambients. For example, if we want to move a running application from work to home, the application must be removed from an enclosing (work) ambient and inserted in a different enclosing (home) ambient.

- Each ambient has a collection of local running processes.

  A local process of an ambient is one that is contained in the ambient but not in any of its subambients. These “top level” local processes have direct control of the ambient, and in particular they can instruct the ambient to move. In contrast, the local processes of a subambient have no direct control on the parent ambient: this helps guaranteeing the integrity of the parent.

- Each ambient moves as a whole with all its subcomponents.

  The activity of a single local process may, by causing movement of its parent, influence the location, and therefore the activity, of other local processes and subambients. For example, if we move a laptop and reconnect it to a different network, then all the threads, address spaces, and file systems within it move accordingly and automatically, and have to cope with their new surrounding. Agent mobility is a special case of ambient mobility, since agents are usually single-threaded. Ambients, like agents, automatically carry with them a collection of private data as they move.

- Each ambient has a name.

  The name of an ambient is used to control access (entry, exit, communication, etc.). In a realistic situation the true name of an ambient would be guarded very closely, and only specific capabilities based on the name would be handed out.

3.3 Ideas for Wide Area Languages

Ambients represent our understanding of the fundamental properties of mobile computation over wide area networks. Our final goal, though, is to program the Internet in some convenient high-level language. Therefore, we aim to find programming constructs that are semantically compatible with the ambient principles, and consequently with wide area networks.

These compatibility requirements include (A) *WAN-soundness*: a wide area network language cannot adopt primitives that entail action-at-a-distance, continued connectivity, global consensus, or security bypasses, and (B) *WAN-completeness*: a wide area network language must be able to express the behavior of web surfers and of mobile agents and users, and of any other entities that routinely roam those networks.

More specifically, we believe the following are necessary ingredients of wide area languages.

- ***Naming***. Names are symbolic ways of referring to entities across a barrier. Names are detached from their corresponding entities; one may possess a name without having immediate access to any entity of that name. To enable mobility and disconnected operation, all entities across a barrier should be denoted by names, not by “hard” pointers.
- ***Migration***. Active hardware and software components should be able to migrate. Migration of certain active hardware components is possible today, but the ability to automatically disconnect and reconnect those components to surrounding (possibly multiple) networks is not currently available. Migration of active software components is even harder, typically for lack of system facilities for migrating live individual threads and groups of threads.
- ***Dynamic connectivity***. A wide area network cannot be started or stopped all at once. Therefore, it is necessary to dynamically connect components. This is contrary to the current prominence in programming languages of static binding, static module composition, and static linking. The ambient calculus provides an example of a novel mixture of ordinary static scoping of names (which enables typechecking) with dynamic binding of operations to names (which enables dynamic linking).
- ***Communication***. Communication on wide area networks must in general be asynchronous. However, local communication (within or even across a single barrier) can usefully be synchronous. Moreover, in the presence of mobility, it is necessary to have some level of synchronization between communication and movement operations. This remains an interesting design area for mobile languages.
- ***Security***. Security abstractions should be provided at the programming-language level, that is, above the fundamental cryptographic primitives. Programmers need to operate with reliable high-level abstractions, otherwise subtle security loopholes can creep in. We believe that barriers are one such high-level security abstraction, which can be supported by programming constructs that can be mechanically analyzed (e.g., via type systems [10]).

Summary

The ambient semantics naturally suggests unusual programming constructs that are well-suited for wide area computation. The combination of mobility, security, communication, and dynamic binding issues has not been widely explored yet at the language-design level, and certainly not within a unifying semantic paradigm. We hope our unifying foundation will facilitate the design of such new languages.

Part II: Ambient Calculus

4 Mobility

We describe a minimal calculus of ambients that includes only mobility primitives. Still, this calculus is already quite expressive. In Section 5 we then introduce communication primitives that allow us to write more natural examples.

4.1 Mobility Primitives

We first introduce a calculus in its entirety, and then we comment on the individual constructions. The syntax of the calculus is defined in the following table. The main syntactic categories are processes (including both ambients and agents that execute actions) and capabilities.

Table 1: Mobility Primitives

n	names
$P,Q ::=$	processes
$(\nu n)P$	restriction
$\mathbf{0}$	inactivity
$P \mid Q$	composition
$!P$	replication
$n[P]$	ambient
$M.P$	action
$M ::=$	capabilities
$in\ n$	can enter n
$out\ n$	can exit n
$open\ n$	can open n

Table 2: Free names

$fn((\nu n)P)$	$\triangleq$	$fn(P) - \{n\}$	$fn(in\ n)$	$\triangleq$	$\{n\}$
$fn(\mathbf{0})$	$\triangleq$	$\varnothing$	$fn(out\ n)$	$\triangleq$	$\{n\}$
$fn(P \mid Q)$	$\triangleq$	$fn(P) \cup fn(Q)$	$fn(open\ n)$	$\triangleq$	$\{n\}$
$fn(!P)$	$\triangleq$	$fn(P)$			
$fn(n[P])$	$\triangleq$	$\{n\} \cup fn(P)$			
$fn(M.P)$	$\triangleq$	$fn(M) \cup fn(P)$			

We write $P\{n\leftarrow m\}$ for the substitution of the name m for each free occurrence of the name n in the process P. Similarly for $M\{n\leftarrow m\}$.

Table 3: Syntactic conventions

$(\nu n)P \mid Q$	is read	$((\nu n)P) \mid Q$	
$!P \mid Q$	is read	$(!P) \mid Q$	
$M.P \mid Q$	is read	$(M.P) \mid Q$	
$(\nu n_1...n_m)P$	$\triangleq$	$(\nu n_1)...(\nu n_m)P$	
$n[]$	$\triangleq$	$n[\mathbf{0}]$	
M	$\triangleq$	$M.\mathbf{0}$	(where appropriate)

The first four process primitives (restriction, inactivity, composition and replication) are commonly found in process calculi. To these we add ambients, $n[P]$, and the exercise of capabilities, $M.P$. Next we discuss these primitives in detail.

4.2 Explanations

We begin by introducing the semantics of ambients informally. A reduction relation $P \longrightarrow Q$ describes the evolution of a process P into a new process Q.

Restriction

The restriction operator:

$$(\nu n)P$$

creates a new (unique) name n within a scope P. The new name can be used to name ambients and to operate on ambients by name.

As in the π-calculus [35], the (νn) binder can float outward as necessary to extend the scope of a name, and can float inward when possible to restrict the scope. Unlike the π-calculus, the names that are subject to scoping are not channel names, but ambient names.

The restriction construct is transparent with respect to reduction; this is expressed by the following rule:

$$P \longrightarrow Q \;\Rightarrow\; (\nu n)P \longrightarrow (\nu n)Q$$

Inaction

The process:

$$\mathbf{0}$$

is the process that does nothing. It does not reduce.

Parallel

Parallel execution is denoted by a binary operator that is commutative and associative:

$$P \mid Q$$

It obeys the rule:

$$P \longrightarrow Q \;\Rightarrow\; P \mid R \longrightarrow Q \mid R$$

This rule directly covers reduction on the left branch; reduction on the right branch is obtained by commutativity.

Replication

Replication is a technically convenient way of representing iteration and recursion. The process:

$$!P$$

denotes the unbounded replication of the process P. That is, $!P$ can produce as many parallel replicas of P as needed, and is equivalent to $P \mid !P$. There are no reduction rules for $!P$; in particular, the term P under ! cannot begin to reduce until it is expanded out as $P \mid !P$.

Ambients

An ambient is written:

$$n[P]$$

where n is the name of the ambient, and P is the process running inside the ambient.

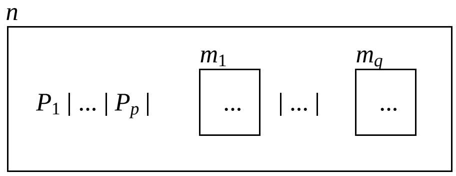
In $n[P]$, it is understood that P is actively running, and that P can be the parallel composition of several processes. We emphasize that P is running even when the surrounding ambient is moving. Running while moving may or may not be realistic, depending on the nature of the ambient and of the communication medium through which the ambient moves, but it is consistent to think in those terms. We express the fact that P is running by a rule that says that any reduction of P becomes a reduction of $n[P]$:

$$P \longrightarrow Q \quad \Rightarrow \quad n[P] \longrightarrow n[Q]$$

In general, an ambient exhibits a tree structure induced by the nesting of ambient brackets. Each node of this tree structure may contain a collection of (non-ambient) processes running in parallel, in addition to subambients. We say that these processes are running in the ambient, in contrast to the ones running in subambients. The general shape of an ambient is, therefore:

$$n[P_1 \mid ... \mid P_p \mid m_1[...] \mid ... \mid m_q[...]] \qquad (P_i \neq n_i[...])$$

To emphasize structure we may display ambient brackets as boxes. Then the general shape of an ambient is:



Nothing prevents the existence of two or more ambients with the same name, either nested or at the same level. Once a name is created, it can be used to name multiple ambients. Moreover, $!n[P]$ generates multiple ambients with the same name. This way, for example, one can easily model the replication of services.

Actions and Capabilities

Operations that change the hierarchical structure of ambients are sensitive. In particular such operations can be interpreted as the crossing of firewalls or the decoding of ciphertexts. Hence these operations are restricted by *capabilities*. Thanks to capabilities, an ambient can allow other ambients to perform certain operations without having to reveal its true name. With the communication primitives of Section 5, capabilities can be transmitted as messages.

The process:

$$M.\ P$$

executes an action regulated by the capability M, and then continues as the process P. The process P does not start running until the action is executed. For each kind of capability M we have a specific rule for reducing $M.\ P$. These rules are described below case by case.

We consider three kinds of capabilities: one for entering an ambient, one for exiting an ambient and one for opening up an ambient. Capabilities are obtained from names; given a name m, the capability *in m* allows entry into m, the capability *out m* allows exit out of m and the capability *open m* allows the opening of m. Implicitly, the possession of one or all of these capabilities is insufficient to reconstruct the original name m from which they were extracted.

Entry Capability

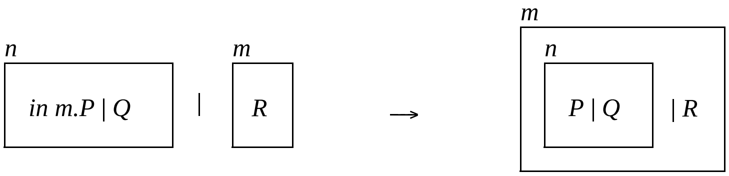
An entry capability, *in m*, can be used in the action:

$$in\ m.\ P$$

which instructs the ambient surrounding *in m. P* to enter a sibling ambient named m. If no sibling m can be found, the operation blocks until a time when such a sibling exists. If more than one m sibling exists, any one of them can be chosen. The reduction rule is:

$$n[\mathit{in}\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$$

Or, by representing ambient brackets as boxes:



If successful, this reduction transforms a sibling *n* of an ambient *m* into a child of *m*. After the execution, the process *in m*. *P* continues with *P*, and both *P* and *Q* find themselves at a lower level in the tree of ambients.

Exit Capability

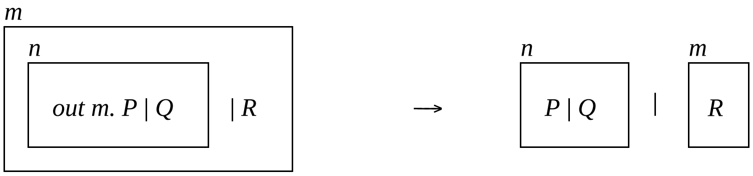
An exit capability, *out m*, can be used in the action:

$$\mathit{out}\ m.\ P$$

which instructs the ambient surrounding *out m*. *P* to exit its parent ambient named *m*. If the parent is not named *m*, the operation blocks until a time when such a parent exists. The reduction rule is:

$$m[n[\mathit{out}\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$$

That is:



If successful, this reduction transforms a child *n* of an ambient *m* into a sibling of *m*. After the execution, the process *in m*. *P* continues with *P*, and both *P* and *Q* find themselves at a higher level in the tree of ambients.

Open Capability

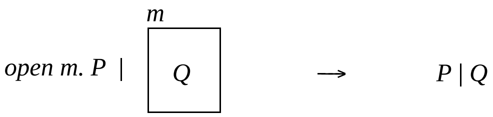
An opening capability, *open m*, can be used in the action:

$$\mathit{open}\ m.\ P$$

This action provides a way of dissolving the boundary of an ambient named *m* located at the same level as *open*, according to the rule:

$$\mathit{open}\ m.\ P \mid m[Q] \longrightarrow P \mid Q$$

That is:



If no ambient *m* can be found, the operation blocks until a time when such an ambient exists. If more than one ambient *m* exists, any one of them can be chosen.

An *open* operation may be upsetting to both *P* and *Q* above. From the point of view of *P*, there is no telling in general what *Q* might do when unleashed. From the point of view of *Q*, its environment is being ripped open. Still, this operation is relatively well-behaved because: (1) the dissolution is initiated by the agent *open m*. *P*, so that the appearance of *Q* at the same level as *P* is not totally unexpected; (2) *open m* is a capability that is given out by *m*, so *m*[*Q*] cannot be dissolved if it does not wish to be (this will become clearer later in the presence of communication primitives).

4.3 Operational Semantics

We now give an operational semantics of the calculus of section 4.1, based on a structural congruence between processes, ≡, and a reduction relation $\longrightarrow$. We have already discussed all the reduction rules, except for one that connects reduction with equivalence. This is a semantics in the style of Milner's reaction relation [34] for the π-calculus, which was itself inspired by the Chemical Abstract Machine of Berry and Boudol [3].

Processes of the calculus are grouped into equivalence classes by the following relation, ≡, which denotes structural congruence (equivalence up to trivial syntactic restructuring).

Table 4: Structural Congruence

$P \equiv P$	(Struct Refl)
$P \equiv Q \;\Rightarrow\; Q \equiv P$	(Struct Symm)
$P \equiv Q,\, Q \equiv R \;\Rightarrow\; P \equiv R$	(Struct Trans)
$P \equiv Q \;\Rightarrow\; (\nu n)P \equiv (\nu n)Q$	(Struct Res)
$P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \;\Rightarrow\; !P \equiv !Q$	(Struct Repl)
$P \equiv Q \;\Rightarrow\; n[P] \equiv n[Q]$	(Struct Amb)
$P \equiv Q \;\Rightarrow\; M.P \equiv M.Q$	(Struct Action)
$P \mid Q \equiv Q \mid P$	(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)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(Struct Res Res)
$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin fn(P)$	(Struct Res Par)
$(\nu n)(m[P]) \equiv m[(\nu n)P] \quad \text{if } n \neq m$	(Struct Res Amb)
$P \mid \mathbf{0} \equiv P$	(Struct Zero Par)
$(\nu n)\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Res)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Repl)

In addition, we identify processes up to renaming of bound names:

$$(\nu n)P \quad = \quad (\nu m)P\{n \leftarrow m\} \quad \text{if } m \notin fn(P)$$

By this we mean that these processes are understood to be identical (for example, by choosing an appropriate representation), as opposed to structurally equivalent.

Note that the following terms are distinct:

$!(\nu n)P \;\not\equiv\; (\nu n)!P$	replication creates new names
$n[P] \mid n[Q] \;\not\equiv\; n[P \mid Q]$	multiple *n* ambients have separate identity

The behavior of processes is given by the following reduction relation. The first three rules are the one-step reductions for *in*, *out* and *open*. The next three rules propagate reductions across scopes, ambient nesting and parallel composition. The final rule allows the use of equivalence during reduction. Finally, $\longrightarrow^*$ is the chaining of multiple reduction steps.

Table 5: Reduction

$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.\ P \mid n[Q] \longrightarrow P \mid Q$	(Red Open)

$P \longrightarrow Q \;\Rightarrow\; (\nu n)P \longrightarrow (\nu n)Q$	(Red Res)
$P \longrightarrow Q \;\Rightarrow\; n[P] \longrightarrow n[Q]$	(Red Amb)
$P \longrightarrow Q \;\Rightarrow\; P \mid R \longrightarrow Q \mid R$	(Red Par)
$P' \equiv P,\ P \longrightarrow Q,\ Q \equiv Q' \;\Rightarrow\; P' \longrightarrow Q'$	(Red $\equiv$)
$\longrightarrow^*$	reflexive and transitive closure of $\longrightarrow$

5 Communication

Although the pure mobility calculus is powerful enough to be Turing-complete, it has no communication or variable-binding operators. Such operators seem necessary, for example, to comfortably encode other formalisms such as the λ-calculus and the π-calculus.

Therefore, we now have to choose a communication mechanism to be used to exchange messages between ambients. The choice of a particular mechanism is to some degree orthogonal to the mobility primitives: many such mechanisms can be added to the mobility core. However, we should try not to defeat with communication the restrictions imposed by capabilities. This suggests that a primitive form of communication should be purely local, and that the transmission of non-local messages should be restricted by capabilities.

To focus our attention, we pose as a goal the ability to encode the asynchronous π-calculus. For this it is sufficient to introduce a simple asynchronous communication mechanism that works locally within a single ambient.

5.1 Communication Primitives

We again start by displaying the syntax of a whole calculus. The mobility primitives are essentially those of section 4, but the addition of communication variables changes some of the details. More interestingly, we add input ($(x).P$) and output ($\langle M \rangle$) primitives and we enrich the capabilities to include paths.

Table 6: Mobility and Communication Primitives

$P,Q ::=$	processes
$(\nu n)P$	restriction
$\mathbf{0}$	inactivity
$P \mid Q$	composition
$!P$	replication
$M[P]$	ambient
$M.P$	capability action
$(x).P$	input action
$\langle M \rangle$	async output action
$M ::=$	messages
x	variable
n	name
$in\ M$	can enter into M
$out\ M$	can exit out of M
$open\ M$	can open M
ε	null
$M.M'$	path

Table 7: Free names (revisions and additions)

$fn(M[P])$	$\triangleq$	$fn(M) \cup fn(P)$	$fn(x)$	$\triangleq$	$\emptyset$
$fn((x).P)$	$\triangleq$	$fn(P)$	$fn(n)$	$\triangleq$	$\{n\}$
$fn(\langle M \rangle)$	$\triangleq$	$fn(M)$	$fn(\varepsilon)$	$\triangleq$	$\emptyset$
			$fn(M.M')$	$\triangleq$	$fn(M) \cup fn(M')$

Table 8: Free variables

$fv((\nu n)P)$	$\triangleq$	$fv(P)$	$fv(x)$	$\triangleq$	$\{x\}$
$fv(\mathbf{0})$	$\triangleq$	$\emptyset$	$fv(n)$	$\triangleq$	$\emptyset$
$fv(P \mid Q)$	$\triangleq$	$fv(P) \cup fv(Q)$	$fv(in\ M)$	$\triangleq$	$fv(M)$
$fv(!P)$	$\triangleq$	$fv(P)$	$fv(out\ M)$	$\triangleq$	$fv(M)$
$fv(M[P])$	$\triangleq$	$fv(M) \cup fv(P)$	$fv(open\ M)$	$\triangleq$	$fv(M)$
$fv(M.P)$	$\triangleq$	$fv(M) \cup fv(P)$	$fv(\varepsilon)$	$\triangleq$	$\emptyset$
$fv((x).P)$	$\triangleq$	$fv(P) - \{x\}$	$fv(M.M')$	$\triangleq$	$fv(M) \cup fv(M')$
$fv(\langle M \rangle)$	$\triangleq$	$fv(M)$			

We write $P\{x \leftarrow M\}$ for the substitution of the capability M for each free occurrence of the variable x in the process P. Similarly for $M\{x \leftarrow M'\}$.

Table 9: New syntactic conventions

$(x).P \mid Q$	is read	$((x).P) \mid Q$

5.2 *Explanations*

Messages

The entities that can be communicated are either names or capabilities. In realistic situations, communication of names should be rather rare, since knowing the name of an ambient gives a lot of control over it. Instead, it should be common to communicate restricted capabilities to allow controlled interactions between ambients.

It now becomes useful to combine multiple capabilities into *paths*, especially when one or more of those capabilities are represented by input variables. To this end we introduce a path-formation operation on capabilities ($M.\ M'$). For example, ($in\ n.\ in\ m$). P is interpreted as $in\ n.\ in\ m.\ P$.

Note also that, for the purpose of communication, we have added names to the collection of messages. A communicated name can be used as a capability to create ambients of that name.

We distinguish between ν-bound names and input-bound variables. Variables can be instantiated with names or capabilities. In practice, we do not need to distinguish these two sorts lexically, but we often use n, m, p, q for names and w, x, y, z for variables.

Ambient I/O

The simplest communication mechanism that we can imagine is local anonymous communication within an ambient (ambient I/O, for short):

$(x).P$	input action
$\langle M \rangle$	async output action

An output action releases a message into the local ether of the surrounding ambient. An input action captures a message from the local ether and binds it to a variable within a scope. We have the reduction:

$$(x).P \mid \langle M \rangle \longrightarrow P\{x \leftarrow M\}$$

This local communication mechanism fits well with the ambient intuitions. In particular, long-range communication, like long-range movement, should not happen automatically because messages may have to cross firewalls.

Still, this simple mechanism is sufficient to emulate communication over named channels, and more generally to provide an encoding of the asynchronous π-calculus [11].

A Syntactic Anomaly

To allow both names and capabilities to be output and input, there is a single syntactic sort that includes both. Hence, a meaningless term of the form $n.\ P$ can arise, for instance, from the process $((x).\ x.\ P) \mid \langle n \rangle$. This anomaly is caused by the desire to denote movement capabilities by variables, as in $(x).\ x.\ P$, and from the desire to denote names by variables, as in $(x).\ x[P]$. We permit $n.\ P$ to be formed, syntactically, in order to make substitution always well defined. In sections 8, 9, and 10, we develop a type system that distinguishes names from capabilities, and that solves this anomaly.

5.3 Operational Semantics

For the extended calculus, the structural congruence relation is defined as in section 4.3, with the understanding that P and M range now over larger classes, and with the addition of the following equivalences:

Table 10: Structural Congruence

$P \equiv Q \;\Rightarrow\; M[P] \equiv M[Q]$	(Struct Amb)
$P \equiv Q \;\Rightarrow\; (x).P \equiv (x).Q$	(Struct Input)
$\varepsilon.P \equiv P$	(Struct ε)
$(M.M').P \equiv M.M'.P$	(Struct .)

We now also identify processes up to renaming of bound variables:

$$(x).P \quad = \quad (y).P\{x\leftarrow y\} \quad \text{if } y \notin fv(P)$$

Finally, we have a new reduction rule for communication:

Table 11: Reduction

$(x).P \mid \langle M \rangle \longrightarrow P\{x\leftarrow M\}$	(Red Comm)

Part III: Equivalence

6 A Theory of Equivalence

Morris-style contextual equivalence [36] (otherwise known as may-testing equivalence [19]) is a standard way of saying that two processes have the same behavior: two processes are contextually equivalent if and only if they admit the same elementary observations whenever they are inserted inside any arbitrary enclosing process. In the setting of the ambient calculus, we shall define contextual equivalence in terms of observing the presence, at the top-level of a process, of an ambient whose name is not restricted.

We say that a process P *exhibits an ambient named n*, and write $P{\downarrow}n$, just if P is a process containing a top-level ambient named n, that is not restricted. We say that a process P *converges to a name n*, and write $P{\Downarrow}n$, just if after some number of reductions, P exhibits an ambient named n. Formally, we define:

$$P{\downarrow}n \quad \triangleq \quad P \equiv (\nu\, m_1...m_i)\,(n[P'] \mid P'') \quad \text{for some } m_1...m_i, P', P'' \text{ where } n \notin \{m_1...m_i\}$$
$$P{\Downarrow}n \quad \triangleq \quad \text{either } P{\downarrow}n \text{ or } P \longrightarrow P' \text{ for some } P' \text{ and } P'{\Downarrow}n$$

Next we define contextual equivalence in terms of the predicate $P{\Downarrow}n$. Let a context $\boldsymbol{C()}$ be a process containing zero or more holes. We write holes as $\mathbf{()}$. For any process P, let $\boldsymbol{C}(P)$ be the process obtained by filling each hole in $\boldsymbol{C}$ with a copy of P. Names free in P may become bound; for example, if $P = n[\langle x\rangle]$ and $\boldsymbol{C()} = (\nu n)(x).\mathbf{()}$, the variable x and the name n have become bound in $\boldsymbol{C}(P) = (\nu n)(x).n[\langle x\rangle]$. Hence, we do not identify contexts up to renaming of bound variables and names.

Let *contextual equivalence* be the relation $P \simeq Q$ defined by:

$$P \simeq Q \quad \triangleq \quad \text{for all } n \text{ and } \boldsymbol{C()},\ \boldsymbol{C}(P){\Downarrow}n \Leftrightarrow \boldsymbol{C}(Q){\Downarrow}n$$

The following two propositions state some basic properties enjoyed by contextual equivalence. Let a relation $\mathcal{R}$ be a precongruence if and only if, for all P, Q, and $\boldsymbol{C()}$, if $P \mathcal{R} Q$ then $\boldsymbol{C}(P)\ \mathcal{R}\ \boldsymbol{C}(Q)$. If, in addition, $\mathcal{R}$ is reflexive, symmetric, and transitive, we say it is a congruence. For example, the structural congruence relation has these properties. Moreover, by a standard argument, so has contextual equivalence:

6-1 Proposition

Contextual equivalence is a congruence.

Structural congruence preserves exhibition of or convergence to a name, and hence is included in contextual equivalence:

6-2 Lemma

Suppose $P \equiv Q$. If $P{\downarrow}n$ then $Q{\downarrow}n$. Moreover, if $P{\Downarrow}n$ then $Q{\Downarrow}n$ with the same depth of inference.

6-3 Proposition

If $\mathrm{P} \equiv \mathrm{Q}$ then $\mathrm{P} \simeq \mathrm{Q}$.

7 Examples

In [27] we present convenient proof techniques for contextual equivalence, and we give a proof of the “perfect firewall equation” described in this section.

7.1 Simple examples

Here are some easy inequivalences:

$p[]$ and $q[]$	are distinguished by the context $\mathbf{()}$
$\mathit{open}\ p.\mathbf{0}$ and $\mathit{open}\ q.\mathbf{0}$	are distinguished by the context $p[n[]] \mid \mathbf{()}$
$\mathit{in}\ p.\mathbf{0}$ and $\mathit{out}\ p.\mathbf{0}$	are distinguished by the context $m[n[\mathbf{()} \mid \mathit{out}\ p.\ \mathit{out}\ m] \mid p[]]$

As in most process calculi, reduction does not imply equivalence:

$$n[] \mid \mathit{open}\ n.\mathbf{0} \longrightarrow \mathbf{0} \quad \text{but} \quad n[] \mid \mathit{open}\ n.\mathbf{0} \not\simeq \mathbf{0}$$

Finally, a general law:

$$(\nu n)(n[] \mid \mathit{open}\ n.P) \simeq (\nu n)P$$

7.2 Example: Perfect firewalls

Consider a process $(\nu n)n[P]$, where n is not free in P. Since the name n is known neither inside the ambient $n[P]$, nor outside it, the ambient $n[P]$ is a “perfect firewall” that neither allows another ambient to enter nor to exit. Such a process could be used to abstractly model an internet firewall named n enclosing a group of machines P, or a sandbox named n enclosing a group of applets P.

The intended property of a perfect firewall can be expressed as an equation; the *Perfect Firewall Equation* below, which expresses the fact that nobody can externally observe a perfect firewall, even though the processes inside the firewall may be active and even cause the firewall to move around.

$$n \notin fn(P) \;\;\Rightarrow\;\; (\nu n)n[P] \simeq \mathbf{0} \qquad\qquad \text{(Alternatively: } (\nu n)n[(\nu n)P] \simeq \mathbf{0}.\text{)}$$

7.3 Example: Perfect encryption

An ambient of the form $k[\langle M\rangle]$ can be used to abstractly model a ciphertext $\{M\}_k$ obtained by encrypting a plaintext M with a key k (we assume that k does not occur in M). The idea is that the key k (or, more precisely the capability *open k*) is needed to access the plaintext M. The intuition that a ciphertext is incomprehensible if one does not know the key can be expressed as the following *Perfect Encryption Equation*. This equation says that any two ciphertexts are indistinguishable (equally unusable) if one does not know their keys. Knowledge of the keys by the context is prevented by the restrictions.

$$(\nu k)k[\langle M\rangle] \simeq (\nu k')k'[\langle M'\rangle]$$

The equation simply follows from the perfect firewall equation, because $(\nu k)k[\langle M\rangle] \simeq \mathbf{0} \simeq (\nu k')k'[\langle M'\rangle]$ if $k \notin fn(M)$ and $k' \notin fn(M')$. It should be compared with the equation $(\nu k)c\langle\{M\}_k\rangle \simeq (\nu k')c\langle\{M'\}_{k'}\rangle$ from the spi calculus [1].

7.4 Example: An agent crossing a firewall

A perfect firewall is not very useful, since it does not let anything pass through. In general we want permeable firewalls that filter traffic according to some security policy. Even these permeable firewalls can be described by equations, although the equations need to become more sophisticated.

In this example, we describe a protocol that allows a “friend” agent to enter a firewall. We assume that the agent and the firewall share some prearranged passwords k, k', k''; these could have been privately generated during a previous visit of the agent to the firewall.

$$\begin{array}{lll} \mathit{Firewall} & \triangleq & (\nu w)w[k[\mathit{out}\ w.\ \mathit{in}\ k'.\ \mathit{in}\ w] \mid \mathit{open}\ k'.\ \mathit{open}\ k''.P] \\ \mathit{Agent} & \triangleq & k'[\mathit{open}\ k.k''[C]] \end{array}$$

Assuming k, k', k'' do not occur in C or P and w does not occur in C, we get the following safety property, expressed as an equation:

$$(\nu k\ k'\ k'')\ (\mathit{Agent} \mid \mathit{Firewall}) \simeq (\nu w)w[C \mid P]$$

Contextual equivalence here means that the agent may successfully cross the firewall (the residual C of the agent may end up inside w). In addition, the agent may successfully cross the firewall under any attack, since equivalence “in every context” includes both “in parallel with any attacking process” and “inside any attacking environment”.

The main caveat is that only high level attacks by ambient calculus processes are covered by this analysis, not low-level attacks such as disassembling the agent in transit. Still, the analysis holds in situation such as applet-to-applet interactions within a trusted server.

Part IV: Types

8 Type Systems

Java has demonstrated the utility of type systems for *mobile code*, and in particular their use and implications for security. Security properties rest on the fact that a well-typed Java program (or the corresponding verified bytecode) cannot cause certain kinds of damage.

We now provide a type system for *mobile computation*, that is, for computation that is continuously active before and after movement. We show that a well-typed mobile computation cannot cause certain kinds of run-time fault: it cannot cause the exchange of values of the wrong kind, anywhere in a mobile system.

In the previous sections we have introduced the (untyped, monadic) *ambient calculus*, a process calculus for mobile computation and mobile devices. That calculus is able to express, via encodings, standard computational constructions such as channel-based communication, functions, and agents [11].

The type system presented here is able to provide typings for those encodings, recovering familiar type systems for processes and functions. In addition, we obtain a type system for mobile agents and other mobile computations [12,10]. The type system is obtained by decorating the untyped calculus with type information.

Our type system tracks the typing of messages exchanged within an ambient. For example, the following system consists of two ambients, named *a* and *b*:

$$a[(x{:}Int).P \mid open\ b] \mid b[in\ a.\ \langle 3\rangle]$$

The ambient named *a* contains a process $(x{:}Int).P$ that is ready to read an integer message into a variable *x* and proceed with *P*, and a process *open b* that is ready to open (dissolve the boundary) of an ambient *b* found within *a*. The ambient named *b* contains a process *in a*. ⟨3⟩ that moves the ambient *b* inside *a* (by executing *in a*) and then outputs the message 3. The ambient *b* is opened after moving into *a*, so the output comes into direct contact with the reading process within *a*. The result is the binding of an integer message to an integer variable, yielding the state:

$$a[P\{\mathrm{x}\leftarrow 3\}]$$

The challenge of the type system is to verify that this exchange of messages is well-typed. Note that in the original system the input and the output were contained in separate locations.

Our ambient calculus is related to earlier distributed variants of the π-calculus, some of which have been equipped with type systems. The type system of Amadio [2] prevents a channel from being defined at more than one location. Sewell's system [40] tracks whether communications are local or non-local, so as to allow efficient implementation of local communication. In Riely and Hennessy's calculus [38], processes need appropriate permissions to perform actions such as migration; a well-typed process is guaranteed to possess the appropriate permission for any action it attempts. Other work on typing for mobile agents includes a type system by De Nicola, Ferrari, and Pugliese [18] that tracks the access rights an agent enjoys at different localities; type-checking ensures that an agent complies with its access rights.

9 The Polyadic Ambient Calculus

We slightly extend the ambient calculus of section 5. In that calculus, communication is based on the exchange of single values. Here we extend the calculus with communication based on tuples of values (polyadic communication), since this simple extension greatly facilitates the task of providing an expressive type system. The process $\langle M_1, ..., M_k\rangle$ represents the output of a tuple of values, with no continuation. The process $(n_1{:}W_1, ..., n_k{:}W_k).P$ represents the input of a tuple of values, with continuation *P*. In addition, we annotate bound variables with type information.

Communication is used to exchange both names and capabilities, which share the same syntactic class *M* of messages. One of the main tasks of our type system is to distinguish the *M*s that are names from the *M*s that are capabilities, so that each is guaranteed to be used in an appropriate context. In general, the type system might distinguish other kinds of messages, such as integer and boolean expressions, but we do not include those in our basic calculus.

Table 12: Polyadic Ambient Calculus

P,Q ::=	processes
$(\nu n{:}W)P$	restriction
$\mathbf{0}$	inactivity
$P \mid Q$	composition
$!P$	replication
$M[P]$	ambient
$M.P$	action
$(n_1{:}W_1, ..., n_k{:}W_k).P$	input
$\langle M_1, ..., M_k\rangle$	async output
M ::=	messages
n	name
$in\ M$	can enter into M
$out\ M$	can exit out of M
$open\ M$	can open M
ε	null path
$M.M'$	composite path

The following tables describe the operational semantics of the extended calculus. The type annotations present in the syntax do not play a role in reduction; they are simply carried along by the reductions and will be explained in the next section.

Table 13: Free names

$fn((\nu n{:}W)P)$	$\triangleq$	$fn(P) - \{n\}$	$fn(n)$	$\triangleq$	$\{n\}$
$fn(\mathbf{0})$	$\triangleq$	$\emptyset$	$fn(in\ M)$	$\triangleq$	$fn(M)$
$fn(P \mid Q)$	$\triangleq$	$fn(P) \cup fn(Q)$	$fn(out\ M)$	$\triangleq$	$fn(M)$
$fn(!P)$	$\triangleq$	$fn(P)$	$fn(open\ M)$	$\triangleq$	$fn(M)$
$fn(M[P])$	$\triangleq$	$fn(M) \cup fn(P)$	$fn(\varepsilon)$	$\triangleq$	$\emptyset$
$fn(M.P)$	$\triangleq$	$fn(M) \cup fn(P)$	$fn(M.M')$	$\triangleq$	$fn(M) \cup fn(M')$
$fn((n_1{:}W_1, ..., n_k{:}W_k).P)$	$\triangleq$	$fn(P) - \{n_1, ..., n_k\}$			
$fn(\langle M_1, ..., M_k\rangle)$	$\triangleq$	$fn(M_1) \cup ... \cup fn(M_k)$			

Table 14: Structural Congruence

$P \equiv P$	(Struct Refl)
$P \equiv Q \;\Rightarrow\; Q \equiv P$	(Struct Symm)
$P \equiv Q,\ Q \equiv R \;\Rightarrow\; P \equiv R$	(Struct Trans)
$P \equiv Q \;\Rightarrow\; (\nu n{:}T)P \equiv (\nu n{:}T)Q$	(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)
$P \equiv Q \;\Rightarrow\; (n_1{:}T_1, ..., n_k{:}T_k).P \equiv (n_1{:}T_1, ..., n_k{:}T_k).Q$	(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)
$(\nu n{:}T)(\nu m{:}U)P \equiv (\nu m{:}U)(\nu n{:}T)P$ if $n \neq m$	(Struct Res Res)
$(\nu n{:}T)(P \mid Q) \equiv P \mid (\nu n{:}T)Q$ if $n \notin fn(P)$	(Struct Res Par)
$(\nu n{:}T)m[P] \equiv m[(\nu n{:}T)P]$ if $n \neq m$	(Struct Res Amb)

$P \mid \mathbf{0} \equiv P$	(Struct Zero Par)
$(\nu n{:}Amb[T])\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Res)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Repl)
$\varepsilon.P \equiv P$	(Struct ε)
$(M.M').P \equiv M.M'.P$	(Struct .)

Table 15: Reduction

$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.\ P \mid n[Q] \longrightarrow P \mid Q$	(Red Open)
$(n_1{:}W_1, ..., n_k{:}W_k).P \mid \langle M_1, ..., M_k\rangle \longrightarrow P\{n_1{\leftarrow}M_1, ..., n_k{\leftarrow}M_k\}$	(Red Comm)
$P \longrightarrow Q \ \Rightarrow\ (\nu n{:}W)P \longrightarrow (\nu n{:}W)Q$	(Red Res)
$P \longrightarrow Q \ \Rightarrow\ n[P] \longrightarrow n[Q]$	(Red Amb)
$P \longrightarrow Q \ \Rightarrow\ P \mid R \longrightarrow Q \mid R$	(Red Par)
$P' \equiv P, P \longrightarrow Q, Q \equiv Q' \ \Rightarrow\ P' \longrightarrow Q'$	(Red ≡)

10 Exchange Types

An ambient is a place where other ambients can enter and exit, and where processes can exchange messages. The first aspect, mobility, is regulated by run-time capabilities and will not be restricted by our type system (but see [10]). The second aspect, communication, is what we now concentrate on.

10.1 Topics of Conversation

Within an ambient, multiple processes can freely execute input and output actions. Since the messages are undirected, it is easily possible for a process to utter a message that is not appropriate for some receiver. The main idea of our type system is to keep track of the *topic of conversation* that is permitted within a given ambient, so that talkers and listeners can be certain of exchanging appropriate messages.

The range of topics is described in the following table by *message types*, *W*, and *exchange types*, *T*. The message types are $Amb[T]$, the type of names of ambients that allow exchanges of type T, and $Cap[T]$, the type of capabilities that when used may cause the unleashing of T exchanges (as a consequence of opening ambients that exchange T). The exchange types are *Shh*, the absence of exchanges, and $W_1 \times ... \times W_k$, the exchange of a tuple of messages with elements of the respective message types. For k=0, the empty tuple type is called **1**; it allows the exchange of empty tuples, that is, it allows pure synchronization. The case k=1 allows any message type to be an exchange type.

Table 16: Types

W ::=	message types
$Amb[T]$	ambient name allowing T exchange
$Cap[T]$	capability unleashing T exchange
T ::=	exchange types
Shh	no exchange
$W_1 \times ... \times W_k$	tuple exchange

For example:

- A quiet ambient: $Amb[Shh]$
- A harmless capability: $Cap[Shh]$

- A synchronization ambient: *Amb*[**1**]
- An ambient that allows the exchange of harmless capabilities: *Amb*[*Cap*[*Shh*]]
- A capability that may unleash the exchange of names of quiet ambients: *Cap*[*Amb*[*Shh*]]

10.2 Intuitions

Before presenting the formal type rules, we discuss the intuitions that lead to them.

Typing of Processes

If a message M has message type W, then $\langle M\rangle$ is a process that outputs (exchanges) W messages. Therefore, we will have a rule stating that:

$$M : W \;\Rightarrow\; \langle M\rangle : W$$

If P is a process that may exchange W messages, then $(x{:}W).P$ is also a process that may exchange W messages. Therefore:

$$P : W \;\Rightarrow\; (x{:}W).P : W$$

The process **0** exchanges nothing, so it naturally has exchange type *Shh*. However, we may also consider **0** as a process that may exchange any type. This is useful when we need to place **0** in a context that is already expected to exchange some given type.

$$\mathbf{0} : T \qquad \text{for any } T$$

If P and Q are processes that may exchange T, then $P \mid Q$ is also such a process. Similarly for $!P$.

$$P : T, Q : T \;\Rightarrow\; P \mid Q : T$$
$$P : T \;\Rightarrow\; !P : T$$

Therefore, by keeping track of the exchange type of a process, T-inputs and T-outputs are tracked so that they match correctly when placed in parallel.

Typing of Ambients

An ambient $n[P]$ is a process that exchanges nothing at the current level, so, like **0**, it can have any exchange type, and can be placed in parallel with any process.

$$n[P] : T \qquad \text{for any } T$$

There needs to be, however, a connection between the type of n and the type of P. We give to each ambient name a type $Amb[T]$, meaning that only T exchanges are allowed in any ambient of that name. Ambients of different names may permit internal exchanges of different types.

$$n : Amb[T], P : T \;\Rightarrow\; n[P] \text{ is well-formed (and can have any type)}$$

By tagging the name of an ambient with the type of exchanges, we know what kind of exchanges to expect in any ambient we enter. Moreover, we can tell what happens when we open an ambient of a given name.

Typing of Open

Tracking the type of I/O exchanges is not enough by itself. We also need to worry about *open*, which might open an ambient and unleash its exchanges inside the surrounding ambient.

If ambients named n permit T exchanges, then the capability *open n* may unleash those T exchanges. We then say that *open n* has a capability type $Cap[T]$, meaning that it may unleash T exchanges when used:

$$n : Amb[T] \quad \Rightarrow \quad open\ n : Cap[T]$$

As a consequence, any process that uses a $Cap[T]$ must be a process that is already willing to participate in exchanges of type T, because further T exchanges may be unleashed.

$$M : Cap[T], P : T \quad \Rightarrow \quad M.P : T$$

The capability types $Cap[T]$ do not keep track of any information concerning *in* and *out* capabilities; only the effect of *open* is tracked.

10.3 Typing Rules

We base our type system on three judgments. The main judgment tracks the exchange type of a process, that is the type of the I/O operations of the process, and of the I/O operations that the process may unleash by opening other ambients.

Judgments

$E \vdash \diamond$	good environment
$E \vdash M : W$	good expression of message type W
$E \vdash P : T$	good process of exchange type T

Based on the discussion in the previous section, we can formalize the type system as described in the following table. Convention: a list of assumptions $E \vdash J_1 \ldots E \vdash J_k$ for k=0 means $E \vdash \diamond$.

Table 17: Rules

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

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

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

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

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

(Exp In)
$$\frac{E \vdash M : Amb[S]}{E \vdash in\ M : Cap[T]}$$

(Exp Out)
$$\frac{E \vdash M : Amb[S]}{E \vdash out\ M : Cap[T]}$$

(Exp Open)
$$\frac{E \vdash M : Amb[T]}{E \vdash open\ M : Cap[T]}$$

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

(Proc Amb)
$$\frac{E \vdash M : Amb[T] \quad E \vdash P : T}{E \vdash M[P] : S}$$

(Proc Res)
$$\frac{E, n{:}Amb[T] \vdash P : S}{E \vdash (\nu n{:}Amb[T])P : S}$$

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

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

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

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

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

- Example: A process that outputs names of quiet ambients:

$$\varnothing \vdash !(\nu n{:}Amb[Shh])\langle n\rangle : Amb[Shh]$$

- Example: A capability that may unleash *S*-exchanges. Note that the *in n* action contributes nothing to the type of the path; only the *open m* action does:

$$\varnothing, n{:}Amb[T], m{:}Amb[S] \vdash in\ n.\ open\ m : Cap[S]$$

The correctness of the type system is expressed by the following proposition:

10-1 Proposition (Subject Reduction)

If $E \vdash P : U$ and $P \longrightarrow Q$ then $E \vdash Q : U$.

□

Certain "run-time error" expressions are allowed in the syntax but are nonsensical because they confuse names with capabilities. Examples are *in n*[*P*], $(\nu n{:}Amb[T])n.P$, and $\langle in\ (in\ n)\rangle$. Such expressions are not initially typeable, and they cannot be produced by well-typed processes because Proposition 10-1 says that the evolution of well-typed processes leads only to well-typed processes.

Part V: Logic

11 Modal Logics for Mobile Ambients

In the course of our ongoing work on mobility, we have often struggled to express precisely certain properties of mobile computations. Informally, these are properties such as "the agent has gone away", "eventually the agent crosses the firewall", "every agent always carries a suitcase", "somewhere there is a virus", or "there is always at most one agent called *n* here". There are several conceivable ways of formalizing these assertions. It is possible to express some of them in terms of equations, as in sections 6 and 7, but this is sometimes difficult or unnatural. It is easier to express some of them as properties of computational traces, but this is very low-level.

Modal logics (particularly, temporal logics) have emerged in many domains as a good compromise between expressiveness and abstraction. In addition, many modal logics support useful computational applications, such as model checking. In our context, it makes sense to talk about properties that hold at particular locations, and it becomes natural to consider *spatial modalities* for properties that hold at a certain location, at some location or at every location. For example, we have the following correspondence between spatial constructs in the ambient calculus and certain formulas:

Processes		*Formulas*	
$\mathbf{0}$	(void)	$\mathbf{0}$	(there is nothing here)
$n[P]$	(location)	$n[\mathcal{A}]$	(there is one thing here)
$P \mid Q$	(composition)	$\mathcal{A} \mid \mathcal{B}$	(there are two things here)

We have a logical constant **0** that is satisfied by the process **0** representing void. We have logical propositions of the form $n[\mathcal{A}]$ (meaning that $\mathcal{A}$ holds at location *n*) that are satisfied by processes of the form $n[P]$ (meaning that process *P* is located at *n*) provided that *P* satisfies $\mathcal{A}$. We have logical propositions of the form $\mathcal{A}' \mid \mathcal{A}''$ (meaning that $\mathcal{A}'$ and $\mathcal{A}''$ hold contiguously) which are satisfied by contiguous processes of the form *P’* | *P”* if *P’* satisfies $\mathcal{A}'$ and *P”* satisfies $\mathcal{A}''$, or vice versa.

Spatial modalities have an intensional flavor that distinguishes our logic from other modal logics for concurrency. Previous work in the area concentrates on properties that are invariant up to strong equivalences such as bisimulation [29,17], while our properties are invariant only up to simple spatial rearrangements. Some of our techniques can be usefully applied to other process calculi, even ones that do not have locations, such as CCS.

Starting from a solid computational intuition, a process calculus, we derive logical inference rules, including the rules of modalities for time, space, and satisfiability, and novel rules for locations and process composition. In [13], based on the satisfaction relation between processes and formulas, we investigate model checking of mobile programs.

12 The Ambient Calculus with Public Names

We consider only ambients having public names; that is we do not deal with name restriction and scope extrusion. Handling of private names in a logic is a very interesting topic, but we leave it for future work.

12.1 Ambients

We modify the basic ambient calculus of section 5. The changes consist in removing name restriction, and in strengthening the definition of structural congruence so that it characterizes the intended equivalence of spatial configurations. We also remove the distinction between names and variables (particularly since we no longer have name binders), and we use a single predicate to characterize free names.

The following table summarizes the syntax of processes. We have separated the process constructs into *spatial* and *temporal*; this is similar to the distinction between static and dynamic constructs in CCS [32].

Table 18: Processes

$P,Q,R ::=$	processes		$M ::=$	messages
$\mathbf{0}$	void	spatial	n	name
$P \mid Q$	composition		*in* M	can enter into M
$!P$	replication		*out* M	can exit out of M
$M[P]$	ambient		*open* M	can open M
$M.P$	capability action	temporal	ε	null
$(n).P$	input action		$M.M'$	path
$\langle M \rangle$	output action			

The set of free names of a process P, written $fn(P)$, is defined as usual; the only binder is in the input action.

12.2 Structural Congruence and Reduction

Structural congruence is used heavily in the logic, as well as in the reduction semantics.

Table 19: Structural Congruence

$P \equiv P$	(Struct Refl)
$P \equiv Q \;\Rightarrow\; Q \equiv P$	(Struct Symm)
$P \equiv Q, Q \equiv R \;\Rightarrow\; P \equiv R$	(Struct Trans)
$P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \;\Rightarrow\; !P \equiv !Q$	(Struct Repl)
$P \equiv Q \;\Rightarrow\; M[P] \equiv M[Q]$	(Struct Amb)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$P \mid \mathbf{0} \equiv P$	(Struct Par Zero)
$!(P \mid Q) \equiv !P \mid !Q$	(Struct Repl Par)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct Repl Zero)

$!P \equiv P \mid !P$	(Struct Repl Copy)
$!P \equiv !!P$	(Struct Repl Repl)
$P \equiv Q \;\Rightarrow\; M.P \equiv M.Q$	(Struct Action)
$P \equiv Q \;\Rightarrow\; (n).P \equiv (n).Q$	(Struct Input)
$\varepsilon.P \equiv P$	(Struct ε)
$(M.M').P \equiv M.M'.P$	(Struct .)

Spatial configurations are ambient configurations consisting only of spatial operators. For example, $a[b[\mathbf{0}] \mid !c[\mathbf{0} \mid \mathbf{0}] \mid !\mathbf{0}]$ is a spatial configuration. These configurations have a natural interpretation as edge-labeled finite-depth trees, where replication introduces infinite branching. The rules for structural congruence are sound and complete for equivalence of these trees. We do not elaborate this further, but it suffices to say that this completeness result motivated the choice of axioms for structural congruence, and particularly the new axioms for replication (which are the same as in Engelfriet's work on the π-calculus [21]).

Table 20: Reduction

$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.\ P \mid n[Q] \longrightarrow P \mid Q$	(Red Open)
$(n).P \mid \langle M \rangle \longrightarrow P\{n \leftarrow M\}$	(Red Comm)
$P \longrightarrow Q \;\Rightarrow\; n[P] \longrightarrow n[Q]$	(Red Amb)
$P \longrightarrow Q \;\Rightarrow\; P \mid R \longrightarrow Q \mid R$	(Red Par)
$P' \equiv P,\ P \longrightarrow Q,\ Q \equiv Q' \;\Rightarrow\; P' \longrightarrow Q'$	(Red ≡)
$\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$	

13 The Logic

In a modal logic, the truth of a formula is relative to a state (or world). In our case, the truth of a *space-time* modal formula is relative to the *here and now*. Each formula talks about the current time, that is, the current state of execution, and the current place, that is, the current location. For example, the formula $n[\mathbf{0}]$ is read: *there is here and now an empty location called n*. The operator $n[\mathcal{A}]$ represents a single step in space, allowing us to talk about the place one step down into n. Another operator, ✧$\mathcal{A}$, allows us to talk about an arbitrary number of steps in space; this is akin to the temporal eventuality operator, $\Diamond\mathcal{A}$.

13.1 Logical Formulas

The syntax of logical formulas is summarized below. This is a modal predicate logic with classical negation. As usual, many standard connectives are interdefinable. The meaning of the formulas will be given shortly in terms of a satisfaction relation. Informally, the first three formulas (true, negation, disjunction) give propositional logic. The next three (void, location, composition) capture spatial configurations, as we discussed. Then we have quantification over names, the two spatial and temporal modalities, and two further operators that we explain later. Quantified variables range only over names: these variables may appear in the location and location adjunct constructs.

The free names of a formula, $fn(\mathcal{A})$, are easily defined since there are no name binders. The free variables of a formula, $fv(\mathcal{A})$, are defined along standard lines: only quantifiers bind variables. A formula $\mathcal{A}$ is closed if $fv(\mathcal{A}) = \emptyset$.

Table 21: Logical Formulas

η	is a name n or a variable x
$\mathcal{A}, \mathcal{B}, \mathcal{C}$ ::=	
$\mathbf{T}$	true
$\neg\mathcal{A}$	negation
$\mathcal{A} \vee \mathcal{B}$	disjunction
$\mathbf{0}$	void
$\eta[\mathcal{A}]$	location
$\mathcal{A} \mid \mathcal{B}$	composition
$\forall x.\mathcal{A}$	universal quantification over names
$\Diamond\mathcal{A}$	sometime modality (temporal)
$✧\mathcal{A}$	somewhere modality (spatial)
$\mathcal{A}@\eta$	location adjunct
$\mathcal{A}\triangleright\mathcal{B}$	composition adjunct

13.2 Satisfaction

The satisfaction relation $P \vDash \mathcal{A}$ means that the process P satisfies the closed formula $\mathcal{A}$. This relation is defined inductively in the following table, where Π is the sort of processes, Φ is the sort of formulas, ϑ is the sort of variables, and Λ is the sort of names. We are very explicit about quantification and sorting of meta-variables because of subtle scoping issues, particularly in the definition of $P \vDash \forall x.\mathcal{A}$. We use the same syntax for logical connectives at the meta-level and object-level, but this is unambiguous.

The meaning of the temporal modality is given by reductions in the operational semantics of the ambient calculus. For the spatial modality, we need the following definition: the relation $P{\downarrow}P'$ indicates that P contains P' within exactly one level of nesting; that is, P' is one step away from P in space, in some downward direction.

$$P{\downarrow}P' \quad \text{iff} \quad \exists n, P''.\ P \equiv n[P'] \mid P''$$

Then, $P{\downarrow}^{*}P'$ is the reflexive and transitive closure of the previous relation, indicating that P contains P' at some nesting level. Note that P' consists of either the top level P, or the entire contents of an enclosed ambient.

Table 22: Satisfaction

$\forall P{:}\Pi.$	$P \vDash \mathbf{T}$		
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \vDash \neg\mathcal{A}$	$\triangleq$	$\neg\ P \vDash \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A},\mathcal{B}{:}\Phi.$	$P \vDash \mathcal{A}\vee\mathcal{B}$	$\triangleq$	$P \vDash \mathcal{A} \vee P \vDash \mathcal{B}$
$\forall P{:}\Pi.$	$\mathrm{P} \vDash \mathbf{0}$	$\triangleq$	$P \equiv \mathbf{0}$
$\forall P{:}\Pi, n{:}\Lambda, \mathcal{A}{:}\Phi.$	$P \vDash n[\mathcal{A}]$	$\triangleq$	$\exists P'{:}\Pi.\ P \equiv n[P'] \wedge P' \vDash \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A},\mathcal{B}{:}\Phi.$	$P \vDash \mathcal{A} \mid \mathcal{B}$	$\triangleq$	$\exists P',P''{:}\Pi.\ P \equiv P'\vert P'' \wedge P' \vDash \mathcal{A} \wedge P'' \vDash \mathcal{B}$
$\forall P{:}\Pi, x{:}\vartheta, \mathcal{A}{:}\Phi.$	$P \vDash \forall x.\mathcal{A}$	$\triangleq$	$\forall m{:}\Lambda.\ P \vDash \mathcal{A}\{x{\leftarrow}m\}$
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \vDash \Diamond\mathcal{A}$	$\triangleq$	$\exists P'{:}\Pi.\ P{\rightarrow}^{*}P' \wedge P' \vDash \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \vDash ✧\mathcal{A}$	$\triangleq$	$\exists P'{:}\Pi.\ P{\downarrow}^{*}P' \wedge P' \vDash \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \vDash \mathcal{A}@n$	$\triangleq$	$n[P] \vDash \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A},\mathcal{B}{:}\Phi.$	$P \vDash \mathcal{A}\triangleright\mathcal{B}$	$\triangleq$	$\forall P'{:}\Pi.\ P' \vDash \mathcal{A} \Rightarrow P\vert P' \vDash \mathcal{B}$

We spell out some of these definitions. A process P satisfies the formula $n[\mathcal{A}]$ if there exists a process P' such that P has the shape $n[P']$ with P' satisfying $\mathcal{A}$. A process P satisfies the formula $\mathcal{A}' \mid \mathcal{A}''$ if there exist processes P' and P'' such that P has the shape $P' \mid P''$ with P' satisfying $\mathcal{A}'$ and P'' satisfying $\mathcal{A}''$. A process P satisfies the formula $\Diamond\mathcal{A}$ if $\mathcal{A}$ holds in the future for some residual *P’* of *P*, where “residual” is defined by $P \rightarrow^{*} P'$. A process *P* satisfies the formula $✧\mathcal{A}$ if $\mathcal{A}$ holds at some sublocation *P’* within *P*, where “sublocation” is defined by $P \downarrow^{*} P'$.

The last two connectives, @ and ▷, can be used to express context/system specifications; they were inspired by the wish to express security properties. A reading of $P \vDash \mathcal{A}@n$ is that *P* (together with its context) manages to satisfy $\mathcal{A}$ even when placed into a location called *n*. A reading of $P \vDash \mathcal{A} \triangleright \mathcal{B}$ is that *P* (together with its context) manages to satisfy $\mathcal{B}$ under any possible attack by an opponent that is bound to satisfy $\mathcal{A}$. Moreover, $P \vDash (\Box\mathcal{A}) \triangleright (\Box\mathcal{A})$ can be interpreted as saying that *P* preserves the invariant $\mathcal{A}$. We will see that these two connectives arise as natural adjuncts to the location and composition connectives.

The following table lists some derived connectives, and it illustrates properties that can be expressed in the logic. The informal meanings can be understood better by expanding out the definitions from the table above.

Table 23: Derived Connectives

$\mathbf{F}$	$\triangleq \neg\mathbf{T}$	false
$\mathcal{A} \wedge \mathcal{B}$	$\triangleq \neg(\neg\mathcal{A} \vee \neg\mathcal{B})$	conjunction
$\mathcal{A} \Rightarrow \mathcal{B}$	$\triangleq \neg\mathcal{A} \vee \mathcal{B}$	implication
$\mathcal{A} \Leftrightarrow \mathcal{B}$	$\triangleq$ $(\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\mathcal{B} \Rightarrow \mathcal{A})$	logical equivalence
$\mathcal{A} \parallel \mathcal{B}$	$\triangleq \neg(\neg\mathcal{A} \mid \neg\mathcal{B})$	decomposition (for every split, one part satisfies $\mathcal{A}$ or the other satisfies $\mathcal{B}$)
$\mathcal{A}^{\forall}$	$\triangleq \mathcal{A} \parallel \mathbf{F}$	every component satisfies $\mathcal{A}$. Ex., each *n* contains just *m*: $(n[\mathbf{T}] \Rightarrow n[m[\mathbf{T}]])^{\forall}$
$\mathcal{A}^{\exists}$	$\triangleq \mathcal{A} \mid \mathbf{T}$	some component satisfies $\mathcal{A}$. Ex., there is an *n* containing an *m*: $n[m[\mathbf{T}]^{\exists}]^{\exists}$
$\exists x.\mathcal{A}$	$\triangleq \neg\forall x.\neg\mathcal{A}$	existential quantification over names. Ex., $\exists x.x[x[\mathbf{T}]]$
$\Box\mathcal{A}$	$\triangleq \neg\Diamond\neg\mathcal{A}$	everytime modality. Ex., *n* is stable: $\Box n[\mathbf{T}]$
$¤\mathcal{A}$	$\triangleq \neg✧\neg\mathcal{A}$	everywhere modality. Ex., there is no *n*: $¤\neg(n[\mathbf{T}]^{\exists})$
$\mathcal{A} \propto \mathcal{B}$	$\triangleq \neg(\mathcal{B} \triangleright \neg\mathcal{A})$	fusion (there is a context satisfying $\mathcal{B}$ that helps ensuring $\mathcal{A}$)
$\mathcal{A} \mid\!\Rightarrow \mathcal{B}$	$\triangleq \neg(\mathcal{A} \mid \neg\mathcal{B})$	fusion adjunct (for every split, if one part satisfies $\mathcal{A}$ the other satisfies $\mathcal{B}$)

Some syntactic conventions: ‘▷’ and the modalities binds more strongly than ‘|’; they all bind more strongly than the standard logical connectives, which have standard precedences. Quantifiers extend to the right as far as possible.

The following is a fundamental property of the satisfaction relation; it states that satisfaction is invariant under structural congruence of processes. In other words, logical formulas can only express properties that are invariant up to structural congruence.

13-1 Proposition (Satisfaction is up to ≡)

$(P \vDash \mathcal{A} \wedge P \equiv P') \Rightarrow P' \vDash \mathcal{A}$

The definition of satisfaction is based heavily on the structural congruence relation. This use of structural congruence may appear arbitrary: other equivalence relations could be used in its place. We have tried to motivate the choice of structural congruence by discussing in Section 12.2 how structural congruence precisely captures the intuition of ambients as spatial configurations. Moreover, structural congruence is easily decidable, which is useful in model-checking applications [13].

14 Validity

In this section, we study valid formulas, valid sequents, and valid logical inference rules. All these are based on the satisfaction relation given in the previous section. Once the definition of satisfaction is fixed, we are basically committed to whatever logic comes out of it. Therefore, it is important to stress that the satisfaction relation appears very natural to us. In particular, the definitions of **0**, $n[\mathcal{A}]$, and $\mathcal{A} \mid \mathcal{B}$ seem inevitable, once we accept that formulas should be able to talk about the tree structure of locations, and that they should not distinguish processes that are surely indistinguishable (up to ≡). The connectives $\mathcal{A}@n$ and $\mathcal{A}\triangleright\mathcal{B}$ have natural security motivations. The modalities $\Diamond\mathcal{A}$ and $✧\mathcal{A}$ talk about process evolution and structure in an undetermined way, which is good for mobility specifications. The rest is classical predicate logic, with the ability to quantify over location names.

Through the satisfaction relation, our logic is based on solid computational intuitions. We should now approach the task of discovering the rules of the logic without preconceptions. As we shall see, what we get has familiar as well as novel aspects.

14.1 The Meaning of Rules

A closed formula is valid if it is satisfied by every process. (For the moment, we consider only validity for closed formulas, i.e., propositional validity.) We use validity for interpreting logical inference rules, as described in the next definition. We use a linearized notation for inference rules, where the usual horizontal bar separating antecedents from consequents is written ‘$\}$’ in-line, and ‘;’ is used to separate antecedents.

Table 24: Validity, Sequents, and Rules

$$\mathbf{vld}(\mathcal{A}) \;\triangleq\; \forall P{:}\Pi.\; P \models \mathcal{A} \;\; (\mathcal{A}, \text{closed, is valid})$$

$$\mathcal{A} \vdash \mathcal{B} \;\triangleq\; \mathbf{vld}(\mathcal{A} \Rightarrow \mathcal{B})$$

$$\mathcal{A}_1 \vdash \mathcal{B}_1; \ldots; \mathcal{A}_n \vdash \mathcal{B}_n \;\}\; \mathcal{A}_0 \vdash \mathcal{B}_0 \;\triangleq\; \mathcal{A}_1 \vdash \mathcal{B}_1 \wedge \ldots \wedge \mathcal{A}_n \vdash \mathcal{B}_n \Rightarrow \mathcal{A}_0 \vdash \mathcal{B}_0 \qquad (n{\geq}0)$$

$$\mathcal{A}_1 \vdash \mathcal{B}_1 \;\{\}\; \mathcal{A}_2 \vdash \mathcal{B}_2 \;\triangleq\; \mathcal{A}_1 \vdash \mathcal{B}_1 \;\}\; \mathcal{A}_2 \vdash \mathcal{B}_2 \;\wedge\; \mathcal{A}_2 \vdash \mathcal{B}_2 \;\}\; \mathcal{A}_1 \vdash \mathcal{B}_1$$

$$\begin{aligned}&\mathcal{A}_1 \vdash \mathcal{B}_1; \ldots; \mathcal{A}_n \vdash \mathcal{B}_n \;\}\; \mathcal{A} \dashv\vdash \mathcal{B} \;\triangleq\\ &\quad \mathcal{A}_1 \vdash \mathcal{B}_1; \ldots; \mathcal{A}_n \vdash \mathcal{B}_n \;\}\; \mathcal{A} \vdash \mathcal{B} \;\wedge\; \mathcal{A}_1 \vdash \mathcal{B}_1; \ldots; \mathcal{A}_n \vdash \mathcal{B}_n \;\}\; \mathcal{B} \vdash \mathcal{A}\end{aligned}$$

We adopt a non-standard formulation of sequents, where each sequent has exactly one assumption and one conclusion: $\mathcal{A} \vdash \mathcal{B}$. Our intention in doing so is to avoid pre-judging the interpretation of the structural operator “,” in standard sequents. In our logic, by taking $\wedge$ on the left and $\vee$ on the right of $\vdash$ as structural operators (i.e., as “,”), all the standard rules of sequent and natural deduction systems with multiple premises/conclusions can be derived. Instead, by taking $\mid$ on the left of $\vdash$ as a structural operator, all the rules of intuitionistic linear logic can be derived. Finally, by taking nestings of $\wedge$ and $\mid$ on the left of $\vdash$ as structural “bunches”, we obtain a bunched logic [37].

14.2 Rules of the Logic

In the sequel, we organize our results into tables of Rules, which are validated in the model, and into tables of Corollaries, which are derived purely logically from the inference rules.

Propositions

The following is a non-standard presentation of the propositional sequent calculus [26], based on our single-assumption single-conclusion sequents. In this presentation, the rules of propositional logic become very symmetrical, and many proofs become more regular, having to consider only single formulas instead of sequences of formulas.

Table 25: Propositional Rules

(A-L)	$\mathcal{A}\wedge(\mathcal{C}\wedge\mathcal{D})\vdash\mathcal{B}\ \{\}\ (\mathcal{A}\wedge\mathcal{C})\wedge\mathcal{D}\vdash\mathcal{B}$	(W-R)	$\mathcal{A}\vdash\mathcal{B}\ \}\ \mathcal{A}\vdash\mathcal{C}\vee\mathcal{B}$
(A-R)	$\mathcal{A}\vdash(\mathcal{C}\vee\mathcal{D})\vee\mathcal{B}\ \{\}\ \mathcal{A}\vdash\mathcal{C}\vee(\mathcal{D}\vee\mathcal{B})$	(Id)	$\}\ \mathcal{A}\vdash\mathcal{A}$
(X-L)	$\mathcal{A}\wedge\mathcal{C}\vdash\mathcal{B}\ \}\ \mathcal{C}\wedge\mathcal{A}\vdash\mathcal{B}$	(Cut)	$\mathcal{A}\vdash\mathcal{C}\vee\mathcal{B};\ \mathcal{A}'\wedge\mathcal{C}\vdash\mathcal{B}'\ \}\ \mathcal{A}\wedge\mathcal{A}'\vdash\mathcal{B}\vee\mathcal{B}'$
(X-R)	$\mathcal{A}\vdash\mathcal{C}\vee\mathcal{B}\ \}\ \mathcal{A}\vdash\mathcal{B}\vee\mathcal{C}$	(**T**)	$\mathcal{A}\wedge\mathbf{T}\vdash\mathcal{B}\ \}\ \mathcal{A}\vdash\mathcal{B}$
(C-L)	$\mathcal{A}\wedge\mathcal{A}\vdash\mathcal{B}\ \}\ \mathcal{A}\vdash\mathcal{B}$	(**F**)	$\mathcal{A}\vdash\mathbf{F}\vee\mathcal{B}\ \}\ \mathcal{A}\vdash\mathcal{B}$
(C-R)	$\mathcal{A}\vdash\mathcal{B}\vee\mathcal{B}\ \}\ \mathcal{A}\vdash\mathcal{B}$	(¬-L)	$\mathcal{A}\vdash\mathcal{C}\vee\mathcal{B}\ \}\ \mathcal{A}\wedge\neg\mathcal{C}\vdash\mathcal{B}$
(W-L)	$\mathcal{A}\vdash\mathcal{B}\ \}\ \mathcal{A}\wedge\mathcal{C}\vdash\mathcal{B}$	(¬-R)	$\mathcal{A}\wedge\mathcal{C}\vdash\mathcal{B}\ \}\ \mathcal{A}\vdash\neg\mathcal{C}\vee\mathcal{B}$

The standard deduction rules of propositional logic, both for the sequent calculus and for natural deduction (interpreting "," as $\wedge$ on the left and $\vee$ on the right), are derivable from the rules in the table.

Composition

The composition rules apply not only to our calculus but also, for example, to CCS.

Table 26: Composition Rules

$(\mid \mathbf{0})$	$\}\ \mathcal{A}\mid\mathbf{0}\dashv\vdash\mathcal{A}$	**0** is nothing
$(\mid \neg\mathbf{0})$	$\}\ \mathcal{A}\mid\neg\mathbf{0}\vdash\neg\mathbf{0}$	if a part is non-**0**, so is the whole
$(\mathrm{A}\mid)$	$\}\ \mathcal{A}\mid(\mathcal{B}\mid\mathcal{C})\dashv\vdash(\mathcal{A}\mid\mathcal{B})\mid\mathcal{C}$	$\mid$ associativity
$(\mathrm{X}\mid)$	$\}\ \mathcal{A}\mid\mathcal{B}\vdash\mathcal{B}\mid\mathcal{A}$	$\mid$ commutativity
$(\mid\vdash)$	$\mathcal{A}'\vdash\mathcal{B}';\ \mathcal{A}''\vdash\mathcal{B}''\ \}\ \mathcal{A}'\mid\mathcal{A}''\vdash\mathcal{B}'\mid\mathcal{B}''$	$\mid$ congruence
$(\mid\vee)$	$\}\ (\mathcal{A}\vee\mathcal{B})\mid\mathcal{C}\vdash\mathcal{A}\mid\mathcal{C}\vee\mathcal{B}\mid\mathcal{C}$	$\mid$-$\vee$ distribution
$(\mid\Vert)$	$\}\ \mathcal{A}'\mid\mathcal{A}''\vdash(\mathcal{A}'\mid\mathcal{B}'')\vee(\mathcal{B}'\mid\mathcal{A}'')\vee(\neg\mathcal{B}'\mid\neg\mathcal{B}'')$	decomposition
$(\mid\rhd)$	$\mathcal{A}\mid\mathcal{C}\vdash\mathcal{B}\ \{\}\ \mathcal{A}\vdash\mathcal{C}\rhd\mathcal{B}$	$\mid$-$\rhd$ adjunction

The converse of |-$\vee$ distribution, $\mathcal{A}\mid\mathcal{C}\vee\mathcal{B}\mid\mathcal{C}\vdash(\mathcal{A}\vee\mathcal{B})\mid\mathcal{C}$, is derivable, and so is a |-$\wedge$ distribution rule, $(\mathcal{A}\wedge\mathcal{B})\mid\mathcal{C}\vdash\mathcal{A}\mid\mathcal{C}\wedge\mathcal{B}\mid\mathcal{C}$. However, the converse of that, namely $\mathcal{A}\mid\mathcal{C}\wedge\mathcal{B}\mid\mathcal{C}\vdash(\mathcal{A}\wedge\mathcal{B})\mid\mathcal{C}$, is not sound. (Take $\mathcal{A}=n[m[\mathbf{T}]]$, $\mathcal{B}=n[p[\mathbf{T}]]$, $\mathcal{C}=n[\mathbf{T}]$, and $P=n[m[]]\mid n[p[]]$; then $P\models\mathcal{A}\mid\mathcal{C}$ and $P\models\mathcal{B}\mid\mathcal{C}$, but $\neg\ P\models(\mathcal{A}\wedge\mathcal{B})\mid\mathcal{C}$.) As a consequence, one cannot always "push | inside $\wedge$" on the left-hand side of a sequent. In particular, after an application of (| $\vdash$) one cannot in general renormalize a sequent to bring $\wedge$'s to the top level.

The decomposition axiom, (| ||), can be used to analyze a composition $\mathcal{A}'\mid\mathcal{A}''$ with respect to arbitrarily chosen $\mathcal{B}'$ and $\mathcal{B}''$. An easy consequence of it is $\neg(\mathcal{A}\mid\mathcal{B})\vdash(\mathcal{A}\mid\mathbf{T})\Rightarrow(\mathbf{T}\mid\neg\mathcal{B})$, which means that if a process cannot be decomposed into parts that satisfy $\mathcal{A}$ and $\mathcal{B}$, but can be decomposed in such a way that a part satisfies $\mathcal{A}$, then it can also be decomposed in such a way that a part does not satisfy $\mathcal{B}$. An even simpler consequence is that $\neg(\mathbf{T}\mid\mathcal{B})\vdash\mathbf{T}\mid\neg\mathcal{B}$, which is one of the few cases in which one can push $\neg$ across |.

The rule (| $\rhd$) states that $\mathcal{A}\rhd\mathcal{B}$ and $\mathcal{A}\mid\mathcal{B}$ are adjuncts. This has a large number of interesting consequences, most of them deriving from the adjunction along standard lines, for example:

Table 27: Some Composition Corollaries

$(\rhd\vdash)$	$\mathcal{A}'\vdash\mathcal{A};\ \mathcal{B}\vdash\mathcal{B}'\ \}\ \mathcal{A}\rhd\mathcal{B}\vdash\mathcal{A}'\rhd\mathcal{B}'$	$(\rhd\rhd)$	$\}\ (\mathcal{A}\rhd\mathcal{B})\mid(\mathcal{B}\rhd\mathcal{C})\vdash\mathcal{A}\rhd\mathcal{C}$
$(\rhd\mid)$	$\}\ (\mathcal{A}\rhd\mathcal{B})\mid\mathcal{A}\vdash\mathcal{B}$	$(\rhd\text{-L})$	$\mathcal{D}\vdash\mathcal{A};\ \mathcal{B}\vdash\mathcal{C}\ \}\ \mathcal{D}\mid(\mathcal{A}\rhd\mathcal{B})\vdash\mathcal{C}$

It is worth pointing out that some composition rules produce interesting interactions between the $\wedge$ and | fragments of the logic. For example, $(\mathcal{A}\mid\mathcal{B})\wedge\mathbf{0}\vdash\mathcal{A}$ is derivable using (| ||) and (| ¬**0**).

Locations

The location rules are very specific to the ambient calculus.

Table 28: Location Rules

$(n[]\,\neg\mathbf{0})$	$\}\ n[\mathcal{A}] \vdash \neg\mathbf{0}$	locations exist
$(n[]\,\neg \mid)$	$\}\ n[\mathcal{A}] \vdash \neg(\neg\mathbf{0} \mid \neg\mathbf{0})$	locations are not decomposable
$(n[]\vdash)$	$\mathcal{A} \vdash \mathcal{B}\ \{\}\ n[\mathcal{A}] \vdash n[\mathcal{B}]$	$n[]$ congruence
$(n[]\wedge)$	$\}\ n[\mathcal{A}]\wedge n[\mathcal{B}] \vdash n[\mathcal{A}\wedge\mathcal{B}]$	$n[]$-$\wedge$ distribution
$(n[]\vee)$	$\}\ n[\mathcal{A}\vee\mathcal{B}] \vdash n[\mathcal{A}]\vee n[\mathcal{B}]$	$n[]$-$\vee$ distribution
$(n[]\,@)$	$n[\mathcal{A}] \vdash \mathcal{B}\ \{\}\ \mathcal{A} \vdash \mathcal{B}@n$	$n[]$-@ adjunction
$(\neg\,@)$	$\}\ \mathcal{A}@n \dashv\vdash \neg((\neg\mathcal{A})@n)$	location adjunct is self-dual

The rule $(n[]\,@)$ states that $\mathcal{A}@n$ and $n[\mathcal{A}]$ are adjuncts. Note that $(n[]\vdash)$ holds in both directions, and that the inverse directions of $(n[]\wedge)$ and $(n[]\vee)$ are derivable; hence, the location fragment of the logic is particularly simple to handle. Some consequences are: $n[\mathcal{A}@n] \vdash \mathcal{A}$, and $\mathcal{A} \dashv\vdash n[\mathcal{A}]@n$, and $\neg n[\mathcal{A}] \dashv\vdash \neg n[\mathbf{T}] \vee n[\neg\mathcal{A}]$.

Time and Space Modalities

The "somewhere" modality was our starting point in developing our logic. We can now investigate its properties.

Table 29: Time and Space Modality Rules

$(\Diamond)$	$\}\ \Diamond\mathcal{A} \dashv\vdash \neg\Box\neg\mathcal{A}$	(✧)	$\}\ \text{✧}\mathcal{A} \dashv\vdash \neg\,\text{¤}\neg\mathcal{A}$
$(\Box\ K)$	$\}\ \Box(\mathcal{A}\Rightarrow\mathcal{B}) \vdash \Box\mathcal{A} \Rightarrow \Box\mathcal{B}$	$(\text{¤}\ K)$	$\}\ \text{¤}(\mathcal{A}\Rightarrow\mathcal{B}) \vdash \text{¤}\mathcal{A} \Rightarrow \text{¤}\mathcal{B}$
$(\Box\ T)$	$\}\ \Box\mathcal{A} \vdash \mathcal{A}$	$(\text{¤}\ T)$	$\}\ \text{¤}\mathcal{A} \vdash \mathcal{A}$
$(\Box\ 4)$	$\}\ \Box\mathcal{A} \vdash \Box\Box\mathcal{A}$	$(\text{¤}\ 4)$	$\}\ \text{¤}\mathcal{A} \vdash \text{¤}\,\text{¤}\mathcal{A}$
$(\Box\ \mathbf{T})$	$\}\,\mathbf{T} \vdash \Box\mathbf{T}$	$(\text{¤}\ \mathbf{T})$	$\}\,\mathbf{T} \vdash \text{¤}\mathbf{T}$
$(\Box\vdash)$	$\mathcal{A}\vdash\mathcal{B}\ \}\ \Box\mathcal{A} \vdash \Box\mathcal{B}$	$(\text{¤}\vdash)$	$\mathcal{A}\vdash\mathcal{B}\ \}\ \text{¤}\mathcal{A} \vdash \text{¤}\mathcal{B}$
$(\text{✧}\Diamond)$	$\}\ \text{✧}\Diamond\mathcal{A} \vdash \Diamond\text{✧}\mathcal{A}$		
$(\Diamond\, n[])$	$\}\ n[\Diamond\mathcal{A}] \vdash \Diamond n[\mathcal{A}]$	$(\text{✧}\, n[])$	$\}\ n[\text{✧}\mathcal{A}] \vdash \text{✧}\mathcal{A}$
$(\Diamond \mid)$	$\}\ \Diamond\mathcal{A} \mid \Diamond\mathcal{B} \vdash \Diamond(\mathcal{A} \mid \mathcal{B})$	$(\text{✧} \mid)$	$\}\ \text{✧}\mathcal{A} \mid \mathcal{B} \vdash \text{✧}(\mathcal{A} \mid \mathbf{T})$

The operators $\Diamond$ and ✧ obey the rules of S4 modalities (the first 5 rules in each column); these follow simply from reflexivity and transitivity of $\longrightarrow^*$ and $\downarrow^*$. These operators, however, are not S5 modalities, that is, $\Diamond\mathcal{A} \vdash \Box\Diamond\mathcal{A}$ is not valid (if $\mathcal{A}$ may happen along some reduction branch, it will not necessarily happen starting from every reduction point), and neither is $\text{✧}\mathcal{A} \vdash \text{¤}\,\text{✧}\mathcal{A}$ (if $\mathcal{A}$ holds in some sublocation, it does not necessarily hold in some sublocation of every sublocation).

The two modalities permute in one direction (somewhere sometime implies sometime somewhere), but the other direction is not sound. (Consider $P = (\mathit{open}\ n.\ m[p[]]) \mid n[]$. Then $P \vDash \Diamond\text{✧}p[\mathbf{0}]$, but $P \nvDash \text{✧}\Diamond p[\mathbf{0}]$). The modalities differ prominently in the way they distribute over compositions and locations, as seen in the bottom rules.

Satisfiability

Validity and satisfiability can be reflected into the logic by means of the $\mathcal{A}^{\mathbf{F}}$ operator (here we use $\mathcal{A}^{\neg}$ for $\neg\mathcal{A}$):

$\mathcal{A}^{\mathbf{F}} \triangleq \mathcal{A}\triangleright\mathbf{F}$	$P \vDash \mathcal{A}^{\mathbf{F}}$ iff $\forall P'{:}\Pi.\ \neg P' \vDash \mathcal{A}$	$\mathcal{A}$ is unsatisfiable
$\mathit{Vld}\,\mathcal{A} \triangleq \mathcal{A}^{\neg\mathbf{F}}$	$P \vDash \mathit{Vld}\,\mathcal{A}$ iff $\forall P'{:}\Pi.\ P' \vDash \mathcal{A}$	$\mathcal{A}$ is valid
$\mathit{Sat}\,\mathcal{A} \triangleq \mathcal{A}^{\mathbf{F}\neg}$	$P \vDash \mathit{Sat}\,\mathcal{A}$ iff $\exists P'{:}\Pi.\ P' \vDash \mathcal{A}$	$\mathcal{A}$ is satisfiable

From the definitions of $\triangleright$ and **F**, we obtain that $P \vDash \mathcal{A}^{\mathbf{F}} \Leftrightarrow (\forall P'{:}\Pi.\ P' \vDash \mathcal{A} \Rightarrow P|P' \vDash \mathbf{F}) \Leftrightarrow (\forall P'{:}\Pi.\ \neg P' \vDash \mathcal{A})$, i.e., iff $\mathcal{A}$ is unsatisfiable independently of P.

One of the main properties of $\mathcal{A}^{\mathbf{F}}$ is that $\mathcal{A} \mid \mathcal{A}^{\mathbf{F}} \vdash \mathbf{F}$, by ($\triangleright$ |). That is, $\mathcal{A}$ cannot be both satisfiable and unsatisfiable. In addition we obtain, from the model, the following rules, from which it is possible to show within the logic that *Vld* and *Sat* obey the rules of S5 modal operators:

Table 30: Satisfiability Rules

($\triangleright\mathbf{F}$ $\neg$)	$\}\ \mathcal{A}^{\mathbf{F}} \vdash \mathcal{A}^{\neg}$	if $\mathcal{A}$ is unsatisfiable then $\mathcal{A}$ is false
($\neg$ $\triangleright\mathbf{F}$)	$\}\ \mathcal{A}^{\mathbf{F}\neg} \vdash \mathcal{A}^{\mathbf{FF}}$	if $\mathcal{A}$ is satisfiable then $\mathcal{A}^{\mathbf{F}}$ is unsatisfiable

Predicates

So far we have considered only propositional validity; when considering quantifiers, we need to extend our notion of validity. If $fv(\mathcal{A})=\{x_1, ..., x_k\}$ are the free variables of $\mathcal{A}$ and $\varphi \in fv(\mathcal{A}) \rightarrow \Lambda$ is a substitution of variables for names, we write $\mathcal{A}_\varphi$ for $\mathcal{A}\{x_1 \leftarrow \varphi(x_1), ..., x_k \leftarrow \varphi(x_k)\}$, and we define:

$$\mathbf{vld}(\mathcal{A}) \ \triangleq\ \forall \varphi \in fv(\mathcal{A}) \rightarrow \Lambda.\ \forall P{:}\Pi.\ P \vDash \mathcal{A}_\varphi$$

This definition of predicate validity generalizes the previous definition of **vld**, which was restricted to the case of $fv(\mathcal{A}) = \emptyset$. It similarly generalizes the definitions of sequents and rules. (Remark: the rules of propositional logic extend to predicate logic.)

Table 31: Quantifier Rules

($\forall$-L)	$\mathcal{A}\{x \leftarrow \eta\} \vdash \mathcal{B}\ \}\ \forall x.\mathcal{A} \vdash \mathcal{B}$	where η is a name or a variable
($\forall$-R)	$\mathcal{A} \vdash \mathcal{B}\ \}\ \mathcal{A} \vdash \forall x.\mathcal{B}$	where $x \notin fv(\mathcal{A})$

As an example, ✧$\forall x.\neg(x[\mathbf{T}]^{\exists})$ is the formula for “somewhere there are no ambients”. Since there are no infinite spatial paths $P_1 \downarrow P_2 \downarrow P_3 \downarrow ...$, we can show in the model that this formula is valid. On the other hand, its temporal dual, “sometime there are no ambients”, $\Diamond\forall x.\neg(x[\mathbf{T}]^{\exists})$, is invalid; for instance, it is not satisfied by $n[]$.

Name Equality

It is possible to encode name equality within the logic in terms of location adjuncts, by taking $\eta = \mu \ \triangleq\ \eta[\mathbf{T}]@\mu$. We obtain that for all $\varphi \in fv(\eta,\mu) \rightarrow \Lambda$ and all $P{:}\Pi$, $P \vDash (\eta = \mu)_\varphi \Leftrightarrow \varphi(\eta) = \varphi(\mu)$. As an example, the following formula means “any two ambients here have different names”, which can be read as a no-spoofing security property:

$$\forall x.\ \forall y.\ x[\mathbf{T}] \mid y[\mathbf{T}] \mid \mathbf{T} \Rightarrow \neg\ x = y$$

15 Conclusions

The global computational infrastructure has evolved in fundamental ways beyond standard notions of sequential, concurrent, and distributed computational models. The notion of *ambients* captures the structure and properties of wide area networks, of mobile computing, and of mobile computation. The ambient calculus [11] formalizes these notions simply and powerfully, supporting reasoning about mobility and security. On this foundation, we can envision new programming methodologies, libraries and languages for wide area computation.

References

[1] Abadi, M. and A.D. Gordon, **A calculus for cryptographic protocols: the spi calculus**. *Proc. of the Fourth ACM Conference on Computer and Communications Security*, 36-47, 1997.

[2] Amadio, R.M., **An asynchronous model of locality, failure, and process mobility**. *Proc. COORDINATION 97*, Lecture Notes in Computer Science 1282, Springer Verlag. 1997.

[3] Berry, G. and G. Boudol, **The chemical abstract machine**. *Theoretical Computer Science* **96**(1), 217-248, 1992.

[4] Bharat, K. and L. Cardelli: **Migratory applications**, *Proc. of the ACM Symposium on User Interface Software and Technology '95*. 133-142. 1995.

[5] Boudol, G., **Asynchrony and the π-calculus.** *Technical Report 1702, INRIA, Sophia-Antipolis*, 1992.

[6] Bracha, G. and S. Toueg, **Asynchronous consensus and broadcast protocols**. *J.ACM* **32**(4), 824-840. 1985.

[7] Cardelli, L., **A language with distributed scope**. *Computing Systems*, **8**(1), 27-59. MIT Press. 1995.

[8] Cardelli, L., **Abstractions for mobile computation**, in *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, Jan Vitek and Christian Jensen (Eds.). Springer. 1999.

[9] Cardelli, L. and R. Davies. **Service combinators for web computing.** *Proc. of the First Usenix Conference on Domain Specific Languages, Santa Barbara*. 1997.

[10] Cardelli, L., Ghelli, G., Gordon, A.D.: **Mobility types for mobile ambients.** ICALP’99. LNCS 1644, 230-239, Springer, 1999.

[11] Cardelli, L. and A.D. Gordon, **Mobile ambients**, in *Foundations of Software Science and Computational Structures*, Maurice Nivat (Ed.), LNCS 1378, Springer, 140-155. 1998

[12] Cardelli, L., and A.D. Gordon, **Types for mobile ambients**. *Proc. 26th Annual ACM Symposium on Principles of Programming Languages*, 79-92. 1999.

[13] Cardelli, L., and A.D. Gordon, **Anytime, anywhere. Modal logics for mobile ambients**. *Proc. 27th Annual ACM Symposium on Principles of Programming Languages*, 2000. (To appear.)

[14] Carriero, N. and D. Gelernter, **Linda in context**. *Communications of the ACM*, **32**(4), 444-458. 1989.

[15] Carriero, N., D. Gelernter, and L. Zuck, **Bauhaus Linda**, in *Object-Based Models and Languages for Concurrent Systems*, P. Ciancarini, O. Nierstrasz and A. Yonezawa (Ed.), Lecture Notes in Computer Science 924, Springer Verlag, 66-76. 1995.

[16] Chandra, T.D., S.Toueg, **Unreliable failure detectors for asynchronous systems.** *ACM Symposium on Principles of Distributed Computing*, 325-340. 1991.

[17] Dam, M.: **Relevance logic and concurrent composition**. LICS'88, 178-185, 1988.

[18] De Nicola, R., G.-L. Ferrari and R. Pugliese, **Locality based Linda: programming with explicit localities**. *Proc. TAPSOFT’97*. Lecture Notes in Computer Science 1214, 712-726, Springer Verlag. 1997.

[19] De Nicola, R., and M.C.B. Hennessy. **Testing equivalences for processes**. *Theoretical Computer Science*, **34**, 83-133, 1984.

[20] Engberg, U.H., Winskel, G.: **Linear logic on Petri nets**. BRICS Report RS-94-3, 1994.

[21] Engelfriet, J.: **A multiset semantics for the π-calculus with replication**. TCS (**153**)65-94, 1996.

[22] Fischer, M.J., N.A. Lynch, and M.S. Paterson, **Impossibility of distributed consensus with one faulty process.** *J.ACM* **32**(2), 374-382. 1985.

[23] Fournet, C. and G. Gonthier, **The reflexive CHAM and the join-calculus**. *Proc. 23rd Annual ACM Symposium on Principles of Programming Languages*, 372-385. 1996.

[24] Fournet, C., G. Gonthier, J.-J. Lévy, L. Maranget, D. Rémy, **A calculus of mobile agents**. *Proc. 7th International Conference on Concurrency Theory (CONCUR'96)*, 406-421. 1996.

[25] Girard, J-Y., Lafont, Y.: **Linear logic and lazy computation**. TAPSOFT 87, LNCS 250 vol 2, 53-66, Springer, 1987.

[26] Girard, J.-Y., Lafont, Y., Taylor, P.: **Proofs and types**. Cambridge University Press, 1989.

[27] Gordon, A.D, and L. Cardelli, **Equational properties of mobile ambients**. *Proc. FOSSACS’99*, Lecture Notes in Computer Science 1578, 212-226, Springer Verlag, 1999.

[28] Gosling, J., B. Joy and G. Steele, **The Java language specification**. Addison-Wesley. 1996.

[29] Hennessy, H., Milner, R.: **Algebraic laws for nondeterminism and concurrency**. JACM, **32**(1)137-161, 1985.

[30] Honda., K. and M. Tokoro, **An object calculus for asynchronous communication.** *Proc. ECOOP’91*, Lecture Notes in Computer Science 521, 133-147, Springer Verlag, 1991.

[31] Lafont, Y.: **The linear abstract machine**. TCS (**59**)157-180, 1988.

[32] Milner, R.: **Flowgraphs and flow algebras**. JACM **26**(4), 1979.

[33] Milner, R., **A calculus of communicating systems**. Lecture Notes in Computer Science 92. Springer Verlag. 1980.

[34] Milner, R., **Functions as processes**. *Mathematical Structures in Computer Science* **2**, 119-141. 1992.

[35] Milner, R., J. Parrow and D. Walker, **A calculus of mobile processes, Parts 1-2**. *Information and Computation*, **100**(1), 1-77. 1992

[36] Morris, J.H., **Lambda-calculus models of programming languages.** Ph.D. Thesis, MIT, Dec 1968.

[37] O'Hearn, P.W., Pym, D.: **The logic of bunched implications**. Bulletin of Symbolic Logic. To appear, 1999.

[38] Riely, J., and M. Hennessy, **A typed language for distributed mobile processes**. *Proc. 25th Annual ACM Symposium on Principles of Programming Languages*, 378-390, 1998.

[39] Sander, A. and C. F. Tschudin, **Towards mobile cryptography**, *ICSI technical report 97-049*, November 1997. *Proc. IEEE Symposium on Security and Privacy*, Spring 1998.

[40] Sewell, P., **Global/local subtyping and capability inference for a distributed π-calculus**. *Proc. ICALP'98*. Lecture Notes in Computer Science 1443, 695-706, Springer, 1998.

[41] Stamos, J.W. and D.K. Gifford, **Remote evaluation**. *ACM Transactions on Programming Languages and Systems* **12**(4), 537-565. 1990.

[42] Urquhart, A.: **Semantics for relevant logics.** Journal of Symbolic Logic **37**(1)159-169, 1972.

[43] White, J.E., **Mobile agents**. In *Software Agents*, J. Bradshaw, ed. The MIT Press. 1996.