UNIVERSITY OF SUSSEX

COMPUTER SCIENCE

Type-Safe Execution of Mobile Agents in Anonymous Networks

Matthew Hennessy and James Riely

Report 3/98 May 1998

Computer Science
School of Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH

ISSN 1350–3170

Type-Safe Execution of Mobile Agents in Anonymous Networks

MATTHEW HENNESSY AND JAMES RIELY

ABSTRACT. We present a *partially-typed* semantics for Dπ, a distributed π-calculus. The semantics is designed for *open* distributed systems in which some sites may harbor malicious agents. Nonetheless, the semantics guarantee traditional type-safety properties at "good" locations by using a mixture of static and dynamic type-checking.

The run-time semantics is built on the model of an *anonymous network* where the source of incoming agents is unknowable. To counteract possible misuse of resources all sites keep a record of local resources against which incoming agents are dynamically typechecked.

1 Introduction

In [7] we presented a type system for controlling the use of resources in a distributed system. The type system guarantees that resource access is always *safe*, in the sense that, for example, integer channels are always used with integers and boolean channels are always used with booleans. The type system of [7], however, requires that all agents in the system be well-typed. In open systems, such as the internet, such global properties are impossible to verify. In this paper, we present a type system for *partially typed* networks, where only a subset of agents are assumed to be well typed.

This notion of partial typing is presented using the language Dπ, from [7]. In Dπ mobile agents are modeled as *threads*, using a thread language based on the π-calculus. Threads are *located*, carrying out computations at *locations* or *sites*. Located threads, or *agents* interact using local *channels*, or *resources*.

In an open system, not all sites are necessarily benign. Some sites may harbor malicious agents that do not respect the typing rules laid down for the use of resources. For example, consider the system

$$\begin{array}{l} k[\![(\nu c{:}\mathsf{chan}\langle\mathsf{int}\rangle)\ m :: a!\langle c\rangle]\!] \\ \mid\ m[\![a?(z[x])\ z :: x!\langle \mathsf{t}\rangle]\!] \end{array}$$

consisting of two sites k and m. The first generates a new local channel c for transmitting integers and makes it known to the second site m, by sending it along the channel a local to m. In a benign world k could assume that any mobile agent that subsequently migrates from m to k would only use this new channel

Research funded by CONFER II and EPSRC project GR/K60701.
Email: {matthewh,jamesri}@cogs.susx.ac.uk

to transmit integers. However in an insecure world m may not play according to the rules; in our example it sends an agent to k which misuses the new resource by sending the boolean value $\mathtt{t}$ along it.

In this paper we formalize one strategy that sites can use to protect themselves from such attacks. The strategy makes no assumptions about the security of the underlying network. For example, it is not assumed that the source of a message (or agent) can be reliably determined. We refer to such networks as *anonymous networks*.

In the presence of anonymous networks a reasonable strategy for sites is based on *paranoia*. Since the source of messages cannot be determined it is impossible to distinguish messages from potentially "trusted" sites; thus no site can be trusted. To protect itself, a site must bar entry of any mobile agent that cannot be proven to use local resources as intended.

The requirement that incoming agents be checked entails that the runtime semantics is of necessity more complicated than that of [7]. Each location must retain information about its local resources against which the incoming agents are checked. This information is encapsulated as a location type, giving the names of the resources available locally, together with their types; agent checking then amounts to a form of *local runtime typechecking*.

The paper proceeds as follows. In the next section we review the language Dπ of [7] and recall its *standard* semantics. While the paper is self contained, we rely on [7] for examples and motivation for Dπ. This is followed, in Section 3, by a formal description of the modified run-time semantics, where sites retain local information against which incoming agents are checked.

In Section 4, we discuss the effectiveness of this runtime strategy; we show that *good sites* cannot be harmed by *bad* sites, in the sense that local resources can not be misused at good sites, even in systems which may contain malicious sites. This is formalized in terms of a static typing system, for which we prove Subject Reduction and Type Safety theorems. The paper ends with a critique of both the run-time strategy and the typing system.

2 The Language

We present the syntax and standard semantics of Dπ. For a full treatment of the language, including many examples, see [7]. The language is a simplification and refinement of that introduced in [12].

2.1 Syntax

The syntax is given in Table 1. We defer the discussion of types, T, to Section 2.3. The syntax is parameterized with respect to the following syntactic sets, which we assume to be disjoint:

TABLE 1 Syntax

$$
\begin{array}{rrcl}
\textit{Id}: & u, v, w & ::= & e \mid x \\
\textit{Val}: & U, V & ::= & i \mid \mathsf{bv} \mid u \mid w[u_1, .., u_n] \mid (U_1, .., U_n) \\
\textit{Pat}: & X, Y & ::= & x \mid z[x_1, .., x_n] \mid (X_1, .., X_n)
\end{array}
$$

$$
\begin{array}{rrcl}
\textit{Thread}: & p, q, r & ::= & \mathsf{nil} \mid u!\langle V\rangle\, p \mid u?(X{:}\mathrm{T})\, q \mid *p \\
& & \mid & u :: p \mid \mathsf{if}\ U = V\ \mathsf{then}\ p\ \mathsf{else}\ q \\
& & \mid & p \mid q \mid (\nu e{:}\mathrm{T})\, p \\
\textit{System}: & P, Q, R & ::= & \mathsf{nil} \mid \ell[\![p]\!] \mid P \mid Q \mid (\nu_\ell e{:}\mathrm{T})\, P
\end{array}
$$

- *Var*, of *variables*, ranged over by *x*-*z*,
- *Name*, of *names*, ranged over by *a*-*m*,
- *Int*, of *integers*, ranged over by i, and
- $\mathit{Bool} = \{\mathsf{t}, \mathsf{f}\}$, of *booleans*, ranged over by bv.

We use *u*-*w* to range over the set of *identifiers*, $\mathit{Id} = \mathit{Var} \cup \mathit{Name}$. We typically use the names *a*-*d* to refer to channels and *k*-*m* to refer to locations, although the distinction is formally imposed by the type system. We use e to refer to names that might be of either type.

The main syntactic categories of the language are as follows:

- *Threads*, *p*-*r*, are (almost) terms of the ordinary polyadic π-calculus [9]. The thread language includes the static combinators for composition '$\mid$' and typed restriction '$(\nu e{:}\mathrm{T})$', as well as constructs for movement '$\ell :: p$', output '$u!\langle V\rangle\, p$', typed input '$u?(X{:}\mathrm{T})\, q$', (mis)matching '$\mathsf{if}\ U = V\ \mathsf{then}\ p\ \mathsf{else}\ q$' and iteration '$*p$'.
- *Agents*, $\ell[\![p]\!]$, are located threads.
- *Systems*, *P*-*R*, are collections of agents combined using the static combinators '$\mid$' and '$(\nu_\ell e{:}\mathrm{T})$'.

The output and input constructs make use of syntactic categories for *values*, *U*-*V*, and *patterns*, *X*-*Y*, respectively. Values include variables, names, base values, and tuples of these. A value of the form $w[u_1, .., u_n]$ includes a location w and a collection of channels $u_1, .., u_n$ allocated at w. Patterns, *X*-*Y*, provide destructors for each type. To be well-formed, we require that patterns be linear, *i.e.* each variable appear at most once.

As an example of a system, consider the term:

$$\ell[\![p]\!] \mid (\nu_\ell a{:}\mathrm{A})\, (\ell[\![q]\!] \mid k[\![r]\!])$$

This system contains three agents, $\ell[\![p]\!]$, $\ell[\![q]\!]$ and $k[\![r]\!]$. The first two agents are running at location ℓ, the third at location k. Moreover q and r share a private

channel a of type A, allocated at ℓ and unknown to p.

Unlike [4, 6], agents are relatively lightweight in Dπ. They are single-threaded and can be freely split and merged using structural rules and communication. As such, they are unnamed.

NOTATION. We adopt several notational conventions, as in [7].

- In the concrete syntax, "move" has greater binding power than composition. Thus $\ell :: p \mid q$ should be read $(\ell :: p) \mid q$. We adopt several standard abbreviations. For example, we routinely drop type annotations when they are not of interest. We omit trailing occurrences of nil and often denote tuples and other groups using a tilde. For example, we write $\widetilde{a}$ instead of $(a_1, .., a_n)$ and $(\nu\widetilde{e}{:}\widetilde{\mathrm{T}})\,p$ instead of $(\nu e_1{:}\mathrm{T}_1)..(\nu e_n{:}\mathrm{T}_n)\,p$. We also write 'if $U = V$ then p' instead of 'if $U = V$ then p else nil' and 'if $U \neq V$ then q' instead of 'if $U = V$ then nil else q.'
- We assume the standard notion of *free* and *bound* occurrences of variables and names in systems and threads. The variables in the pattern X are bound by the input construct $u?(X)\,q$, the scope is q. The name e is bound by the restrictions $(\nu_\ell e{:}\mathrm{T})\,P$ and $(\nu e{:}\mathrm{T})\,p$, the scopes are P and p, respectively. A term with no free variables is *closed*. The functions $\mathrm{fn}(P)$ and $\mathrm{fid}(P)$ return respectively the sets of free names and free identifiers occurring in P.
- We also assume a standard notion *substitution*, where $P\{{}^{u}/_{x}\}$ denotes the capture-avoiding substitution of u for x in P. The notation $P\{{}^{V}/_{X}\}$ generalizes this in an obvious way as a sequence of substitutions. For example, $P\{{}^{\ell[a]}/_{z[x]}\} = P\{{}^{\ell}/_{z}\}\{{}^{a}/_{x}\}$.
- In the sequel we identify terms up to renaming of bound identifiers. □

2.2 Standard Reduction

The standard reduction semantics is given in Table 2. The structural equivalence ($P \equiv Q$) and reduction relation ($P \longrightarrow P'$) both relate *closed* system terms. The structural equivalence is defined to be the least equivalence relation that is closed under composition and restriction, satisfies the monoid laws for composition,[1] and in addition satisfies the axioms given in Table 2.

The structural rules allow for agent splitting $\ell[\![p \mid q]\!] \equiv \ell[\![p]\!] \mid \ell[\![q]\!]$ and garbage collection $\ell[\![\mathsf{nil}]\!] \equiv \mathsf{nil}$. Note that when a new name is lifted out of an agent, using the structural rule $(\mathsf{s\text{-}new})$, the system-level restriction records the name of the location which allocated the name $\ell[\![(\nu e{:}\mathrm{T})\,p]\!] \equiv (\nu_\ell e{:}\mathrm{T})\,\ell[\![p]\!]$; these location tags are used only for static typing.

[1] The monoid laws are: $P \mid \mathsf{nil} \equiv P$, $P \mid Q \equiv Q \mid P$, and $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$.

TABLE 2 Standard Reduction

Structural equivalence:

$$
\begin{array}{rrcl}
(\textsf{s-nil}) & \ell[\![\textsf{nil}]\!] & \equiv & \textsf{nil} \\
(\textsf{s-split}) & \ell[\![p \mid q]\!] & \equiv & \ell[\![p]\!] \mid \ell[\![q]\!] \\
(\textsf{s-itr}) & \ell[\![*p]\!] & \equiv & \ell[\![p]\!] \mid \ell[\![*p]\!] \\
(\textsf{s-new}) & \ell[\![(\nu e{:}\mathrm{T})\, p]\!] & \equiv & (\nu_\ell e{:}\mathrm{T})\, \ell[\![p]\!] \\
(\textsf{s-extr}) & Q \mid (\nu_\ell e{:}\mathrm{T})\, P & \equiv & (\nu_\ell e{:}\mathrm{T})\, (Q \mid P) \quad \text{if } e \notin \mathrm{fn}(Q)
\end{array}
$$

Reduction:

$$
\begin{array}{rl}
(\textsf{r-move}) & \ell[\![k :: p]\!] \longrightarrow k[\![p]\!] \\
(\textsf{r-comm}) & \ell[\![a!\langle V\rangle\, p]\!] \mid \ell[\![a?(X)\, q]\!] \longrightarrow \ell[\![p]\!] \mid \ell[\![q\{^V/_X\}]\!] \\
(\textsf{r-eq}_1) & \ell[\![\textsf{if } U = U \textsf{ then } p \textsf{ else } q]\!] \longrightarrow \ell[\![p]\!] \\
(\textsf{r-eq}_2) & \ell[\![\textsf{if } U = V \textsf{ then } p \textsf{ else } q]\!] \longrightarrow \ell[\![q]\!] \quad \text{if } U \neq V \\
(\textsf{r-new}) & \dfrac{P \longrightarrow P'}{(\nu_\ell e{:}\mathrm{T})\, P \longrightarrow (\nu_\ell e{:}\mathrm{T})\, P'} \\
(\textsf{r-str}) & \dfrac{P \longrightarrow P'}{R \mid P \longrightarrow R \mid P'} \qquad \dfrac{P \equiv Q \longrightarrow Q' \equiv P'}{P \longrightarrow P'}
\end{array}
$$

Most of the rules of the reduction relation are taken directly from the π-calculus, with a few changes to accommodate the fact that agents are explicitly located. The main rule of interest here is (r-move),

$$\ell[\![k :: p]\!] \longrightarrow k[\![p]\!]$$

which states that an agent located at ℓ can move to k using the move operator $k :: p$. This is the only rule that varies significantly between the standard semantics and the semantics for open systems defined later. Note that in (r-comm), communication is purely local:

$$\ell[\![a!\langle V\rangle\, p]\!] \mid \ell[\![a?(X)\, q]\!] \longrightarrow \ell[\![p]\!] \mid \ell[\![q\{^V/_X\}]\!]$$

As an example, suppose that we wish to write a system of two agents, one at k and one at ℓ. The agent at k wishes to send a fresh integer channel a, located at k, to the other agent using the channel b, located at ℓ. This system could be written:

$$
\begin{array}{rll}
 & \ell[\![b?(z[x])\, q]\!] \mid k[\![(\nu a)\, (p \mid \ell :: b!\langle k[a]\rangle)]\!] & \\
\equiv & \ell[\![b?(z[x])\, q]\!] \mid (\nu_k a)\, \left(k[\![p]\!] \mid k[\![\ell :: b!\langle k[a]\rangle]\!]\right) & (\textsf{s-new}), (\textsf{s-split}) \\
\longrightarrow & \ell[\![b?(z[x])\, q]\!] \mid (\nu_k a)\, \left(k[\![p]\!] \mid \ell[\![b!\langle k[a]\rangle]\!]\right) & (\textsf{r-move}) \\
\longrightarrow & (\nu_k a)\, \left(\ell[\![q\{^{k[a]}/_{z[x]}\}]\!] \mid k[\![p]\!]\right) & (\textsf{s-extr}), (\textsf{r-comm}), (\textsf{s-nil})
\end{array}
$$

Beside each reduction, we have written the rules used to infer it, omitting (r-str) and (r-new), which are almost always used. Note that after arriving at ℓ, the agent sends the value $k[a]$ rather than simply a. In the type system, this identifies the resource a as *non-local* at ℓ. If a simple resource value, such as a, had been communicated to q, it would have had to have been local to ℓ, rather than k. An example of a process q that uses the received value $z[x]$ is $z :: x!\langle 1\rangle$, which after the communication would become $k :: a!\langle 1\rangle$. This can move to the location k and send the integer 1 on the newly received channel a.

2.3 Types and Subtyping

The purpose of the type system is to ensure proper use of base types, channels and locations. In this paper we use the simple type languages from [7, §4], extended with base types for integers and booleans. We use uppercase Roman letters to range over types, whose syntax is as follows:

$$\begin{array}{rrcl} \textit{TChan}: & \mathrm{A},\mathrm{B},\mathrm{C} & ::= & \mathsf{chan}\langle \mathrm{T}\rangle \\ \textit{TLoc}: & \mathrm{K},\mathrm{L},\mathrm{M} & ::= & \mathsf{loc}\{a_1{:}\mathrm{A}_1, .., a_n{:}\mathrm{A}_n\},\ a_i \text{ distinct} \\ \textit{TVal}: & \mathrm{S},\mathrm{T} & ::= & \mathsf{bool} \mid \mathsf{int} \mid \mathrm{A} \mid \mathrm{K}[\mathrm{B}_1, .., \mathrm{B}_h],\ \mid (\mathrm{T}_1, .., \mathrm{T}_n) \end{array}$$

Types are divided into the following syntactic groups:

- *TChan* of channel types, which specify the type of values communicated over a channel, $\mathsf{chan}\langle \mathrm{T}\rangle$.
- *TLoc* of simple location types, which specify the set of typed channels available at a location, $\mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}\}$.
- *TVal* of value types, which include types for base values, channels, locations and tuples.

In value types, location types have the form $\mathsf{loc}\{a_1{:}\mathrm{A}_1, .., a_n{:}\mathrm{A}_n\}[\mathrm{B}_1, .., \mathrm{B}_h]$. The extended form allows for a certain amount of first-order existential polymorphism. Informally, $\mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}\}[\widetilde{\mathrm{B}}]$ may be read "$\exists\widetilde{x}\colon \mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}},\widetilde{x}{:}\widetilde{\mathrm{B}}\}$", *i.e.* the type of a location which has channels $\widetilde{a}$ of types $\widetilde{A}$ and some (unnamed) channels of types $\widetilde{B}$.

Throughout the text, we drop empty braces when clear from context, writing 'loc' instead of '$\mathsf{loc}\{\,\}[\,]$', 'K' instead of 'K[ ]', and 'u' instead of '$u[\,]$'.

Location types are essentially the same as standard record types, and we identify location types up to reordering of their "fields". Thus $\mathsf{loc}\{a{:}\mathrm{A}, b{:}\mathrm{B}\}[\mathrm{C}] = \mathsf{loc}\{b{:}\mathrm{B}, a{:}\mathrm{A}\}[\mathrm{C}]$. But reordering is not allowed on "abstract" fields. Thus if B and C are different, then $\mathsf{loc}\{a{:}\mathrm{A}\}[\mathrm{B},\mathrm{C}] \neq \mathsf{loc}\{a{:}\mathrm{A}\}[\mathrm{C},\mathrm{B}]$.

The subtyping relation ($\mathrm{T} \leq \mathrm{S}$) is discussed at length in [7]. On base types and channel types there is no nontrivial subtyping; for example, $\mathsf{chan}\langle \mathrm{T}\rangle \leq \mathsf{chan}\langle \mathrm{T}'\rangle$ if and only if $\mathrm{T} = \mathrm{T}'$. On location types (both simple and "existential"), the subtyping relation is similar to that traditionally defined for record or

object types:

$$\mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}, b{:}\mathrm{B}\} \leq \mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}\}$$
$$\mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}, b{:}\mathrm{B}\}[\widetilde{C}] \leq \mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}\}[\widetilde{C}]$$

On tuples, the definition is by homomorphic extension: $\widetilde{\mathrm{S}} \leq \widetilde{\mathrm{T}}$ if $\forall i\colon \mathrm{S}_i \leq \mathrm{T}_i$

2.4 Type Environments

Location types contain the names of the channels known to be defined at a location. To present typing systems for the language in later sections, it is useful to generalize location types to allow the inclusion of *variables* as well as names. Variables are allowed at types int and bool, in addition to channel types A. The resulting types are called *open* location types, $\mathbb{K}$:

$$\begin{array}{rl} \textit{TSimple}: & \mathrm{H}, \mathrm{G} \ ::= \mathsf{int} \mid \mathsf{bool} \mid \mathrm{A} \\ \textit{TOpen}: & \mathbb{K}, \mathbb{L}, \mathbb{M} \ ::= \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{H}}\}, \ u_i \text{ distinct} \end{array}$$

To be well-formed, we require that every *name* in an open location type be associated with a channel type.

The subtyping relation extends directly to open location types:

$$\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{H}}, \widetilde{v}{:}\widetilde{\mathrm{G}}\} \leq \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{H}}\}$$

A type environment, Γ, maps identifiers to open location types. An example of a type environment is:

$$\Gamma = \left\{ \begin{array}{l} k : \mathsf{loc}\,\{a{:}\mathsf{chan}\langle\mathsf{int}\rangle, x{:}\mathsf{int}\} \\ z : \mathsf{loc}\left\{ \begin{array}{l} a{:}\mathsf{chan}\langle\mathsf{loc}[\mathsf{chan}\langle\mathsf{int}\rangle]\rangle \\ y{:}\mathsf{chan}\langle\mathsf{loc}[\mathsf{chan}\langle\mathsf{bool}\rangle]\rangle \end{array} \right\} \end{array} \right\}$$

Here we have two locations, k and z. The first has an integer channel named a and an integer variable x. The second has two channels: a, which communicates (potentially remote) integer channels and y which communicates (potentially remote) boolean channels.

If a type environment contains no variables, we say that it is *closed*. Closed type environments map names to (closed) location types K.

In the typing system of the next section we need some notation for extending (open) location types and for convenience we explain this notation here. The open location $\mathbb{L}$ may be extended using '$\mathbb{L}$, V:T', defined by induction on V:

$$\begin{array}{rl} (\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{H}}\}), v{:}\mathrm{G} = \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{H}}, v{:}\mathrm{G}\} & \text{if } \forall i\colon u_i \neq v \\ \mathbb{L}, w[\widetilde{u}]{:}\mathrm{K}[\widetilde{\mathrm{A}}] = \mathbb{L} & \text{if } \{w, \widetilde{u}\} \text{ disjoint } \mathrm{dom}(\mathbb{L}) \\ \mathbb{L}, \mathsf{bv}{:}\mathsf{bool} = \mathbb{L} & \\ \mathbb{L}, i{:}\mathsf{int} = \mathbb{L} & \\ \mathbb{L}, \widetilde{U}{:}\widetilde{\mathrm{T}} = ((\ldots(\mathbb{L}, U_1{:}\mathrm{T}_1), \ldots), U_n{:}\mathrm{T}_n) & \end{array}$$

TABLE 3 Reduction for Open Systems

$$(\textsf{r-move})\quad \Delta \rhd \ell[\![k :: p]\!] \longrightarrow k[\![p]\!] \quad \text{if } \Delta(k) \vdash p$$

$$(\textsf{r-comm})\quad \Delta \rhd \ell[\![a!\langle V\rangle p]\!] \mid \ell[\![a?(X)\, q]\!] \longrightarrow \ell[\![p]\!] \mid \ell[\![q\{^{V}/_{X}\}]\!]$$

$$(\textsf{r-eq}_1)\quad \Delta \rhd \ell[\![\textsf{if } U = U \textsf{ then } p \textsf{ else } q]\!] \longrightarrow \ell[\![p]\!]$$

$$(\textsf{r-eq}_2)\quad \Delta \rhd \ell[\![\textsf{if } U = V \textsf{ then } p \textsf{ else } q]\!] \longrightarrow \ell[\![q]\!] \quad \text{if } U \neq V$$

$$(\textsf{r-new})\quad \frac{\Delta,\, \ell{:}(\mathrm{L},\, a{:}\mathrm{A}) \rhd P \longrightarrow P'}{\Delta,\, \ell{:}\mathrm{L} \rhd (\nu_\ell a{:}\mathrm{A})\, P \longrightarrow (\nu_\ell a{:}\mathrm{A})\, P'} \qquad \frac{\Delta,\, \ell{:}\mathrm{L},\, k{:}\mathrm{K} \rhd P \longrightarrow P'}{\Delta,\, \ell{:}\mathrm{L} \rhd (\nu_\ell k{:}\mathrm{K})\, P \longrightarrow (\nu_\ell k{:}\mathrm{K})\, P'}$$

$$(\textsf{r-str})\quad \frac{\Delta \rhd P \longrightarrow P'}{\Delta \rhd R \mid P \longrightarrow R \mid P'} \qquad \frac{P \equiv Q \quad \Delta \rhd Q \longrightarrow Q' \quad Q' \equiv P'}{\Delta \rhd P \longrightarrow P'}$$

Therefore the extension of $\mathbb{L}$ by $V{:}\mathrm{T}$ adds only information about *local identifiers* to $\mathbb{L}$. For example:

$$\mathsf{loc}\{d{:}\mathrm{D}\}, (0, x, z[y]){:}(\mathsf{int}, \mathrm{A}, \mathsf{loc}\{c{:}\mathrm{C}\}[\mathrm{B}]) \;=\; \mathsf{loc}\{d{:}\mathrm{D},\, x{:}\mathrm{A}\}$$

Note also that every location type L *is* an open location type, and thus this definition applies to "closed" location types as well.

We use a similar notation for extending type environments: If u is not in the domain of Γ then $\Gamma, u{:}\mathbb{L}$ denotes the new type environment, which is similar to Γ but in addition maps u to the $\mathbb{L}$.

3 Semantics for Open Systems

As we have explained in the introduction, using the standard run-time semantics local resources may be intentionally misused by malicious sites; in the example discussed on page 1 we have seen that the local resource a at location k is deliberately misused by a mobile agent originating at location m. In this section we modify the run-time semantics in a manner which offers protection against such misuse.

We are assuming that agents are working in an anonymous network and the run-time strategy suggested in the introduction is for every site to keep a record of local resources against which all incoming agents are checked. In our formulation this distributed information is collected together as closed type environment and thus reductions of the runtime semantics take the form

$$\Delta \rhd P \longmapsto P'$$

where Δ is a closed type environment and P and P' are systems.

The reduction semantics is given in Table 3. Most of the rules are simple adaptations of the corresponding rules in Table 2. For example, the rules for

TABLE 4 Typing for Values and Threads

Values:

$$(\textsf{t-sit})\ \frac{\mathbb{L} \leq \mathsf{loc}\{u{:}\mathrm{H}\}}{\mathbb{L} \vdash u{:}\mathrm{H}} \qquad (\textsf{t-base})\ \frac{}{\mathbb{L} \vdash n{:}\mathsf{int},\, \mathsf{bv}{:}\mathsf{bool}}$$

$$(\textsf{t-loc})\ \frac{}{\mathbb{L} \vdash u[v_1, .., v_n]{:}\mathrm{K}[\mathrm{A}_1, .., \mathrm{A}_n]} \qquad (\textsf{t-tup})\ \frac{\forall i{:}\ \mathbb{L} \vdash U_i{:}\mathrm{T}_i}{\mathbb{L} \vdash (U_1, .., U_n){:}(\mathrm{T}_1, .., \mathrm{T}_n)}$$

Threads:

$$(\textsf{t-move})\ \frac{}{\mathbb{L} \vdash u :: p} \qquad (\textsf{t-newl})\ \frac{\mathbb{L} \vdash p}{\mathbb{L} \vdash (\nu k{:}\mathrm{K})\, p}$$

$$(\textsf{t-r})\ \frac{\mathbb{L} \vdash u{:}\mathsf{chan}\langle \mathrm{T}\rangle \quad \mathbb{L}, X{:}\mathrm{T} \vdash q}{\mathbb{L} \vdash u?(X{:}\mathrm{T})\, q} \qquad (\textsf{t-newc})\ \frac{\mathbb{L}, a{:}\mathrm{A} \vdash p}{\mathbb{L} \vdash (\nu a{:}\mathrm{A})\, p}$$

$$(\textsf{t-w})\ \frac{\mathbb{L} \vdash u{:}\mathsf{chan}\langle \mathrm{T}\rangle, V{:}\mathrm{T}, p}{\mathbb{L} \vdash u!\langle V\rangle\, p} \qquad (\textsf{t-str})\ \frac{\mathbb{L} \vdash p, q}{\mathbb{L} \vdash \mathsf{nil}, *p, p \,|\, q}$$

$$(\textsf{t-eq})\ \frac{\mathbb{L} \vdash U{:}\mathrm{T}, V{:}\mathrm{T}, p, q}{\mathbb{L} \vdash \mathsf{if}\ U = V\ \mathsf{then}\ p\ \mathsf{else}\ q}$$

local communication and matching of values are essentially as before, as Δ is not consulted for these reductions. There is a minor change in the rule for the restriction operator, because Δ must be augmented to reflect the addition of the new name.

The only significant change from the standard run-time semantics is in the rule for code movement:

$$\Delta \triangleright \ell[\![k :: p]\!] \longmapsto k[\![p]\!] \qquad \text{if } \Delta(k) \vdash p$$

This says that the agent p can move from location ℓ to location k only if p is guaranteed not to misuse the local resources of k, *i.e.* $\Delta(k) \vdash p$. Here p is type-checked dynamically against $\Delta(k)$, which gives the names and types of the resources available at k.

3.1 Runtime Typing

The definition of this runtime local type-checking is given in Table 4. This is a *light weight* typing in that the incoming code is only checked to the extent of its

references to local resources. Thus judgments are of the form

$$\mathbb{L} \vdash p$$

indicating that p can safely run at a location that provides resources as defined in $\mathbb{L}$.

Perhaps the most surprising rule in this light weight type checking is (t-move), which involves no type checking whatsoever. However this is reasonable as an agent such as $\ell :: p$ running at k uses no local resources; it moves immediately to the site ℓ. As a result of this rule notice that reductions of the form

$$\Delta \triangleright m[\![k :: \ell :: p]\!] \longmapsto k[\![\ell :: p]\!]$$

are always allowed, regardless of the information in Δ.

The only significant local checking is carried out by the two rules (t-r), (t-w) which we examine in some detail. The subtlety in the read rule (t-r) is to some extent hidden in the rule for updating location types and this is best explained by example. The rule dictates, for example, that the agent $a?(x{:}\mathsf{chan}\langle \mathrm{T}\rangle)\, q$ can migrate to a location with local resources $\mathbb{L}$ provided:

- a is a local channel of the appropriate type, in this case a channel for communicating values of type $\mathsf{chan}\langle \mathrm{T}\rangle$
- q is locally well-typed with respect to an augmented set of resources, $\mathbb{L}, x{:}\mathsf{chan}\langle \mathrm{T}\rangle$.

However suppose the channel a communicates non-local information; *e.g.* when is $a?(z[x]{:}\mathrm{K}[\mathrm{A}])\, q$ locally well-typed? The rule (t-r) simply demands that, in addition to a having the appropriate local type, q is well-typed with respect to the same set of local resources $\mathbb{L}$. Formally this is because according to the definitions given in Section 2.4 $\mathbb{L}, z[x]{:}\mathrm{K}[\mathrm{A}]$ is simply $\mathbb{L}$. Intuitively this is reasonable since any non-local information received on a will not be used locally and thus it may be ignored.

The rule (t-w) states that agent $a!\langle V\rangle\, p$ is locally well-typed provided

- the continuation p is locally well-typed
- the channel a has an appropriate local type, say $\mathsf{chan}\langle \mathrm{T}\rangle$
- the value to be transmitted V is locally well-typed to be transmitted on a, $\mathbb{L} \vdash V{:}\mathrm{T}$; this means that it must be possible to assign to V the local object type of the channel a, namely T.

Once more there is a subtlety, this time in the local type checking of values. If the value V to be transmitted is a local resource, say a channel name b, then according to the rule (t-sit) b must have the local type T. If, on the other hand, V is a non-local value, say $k[b]$, then locally this is of no interest; according to (t-loc) $k[b]$ can be assigned *any* location type which in effect means that when it

is transmitted locally on a its validity is not checked.

This ends our discussion of runtime local type checking, and of the runtime semantics.

3.2 An Example

As an example consider a system of three locations, k, ℓ and m, with the following distributed type environment, Δ.

$$\Delta = \left\{ \begin{array}{l} k : \mathsf{loc}\{a : \mathsf{chan}\langle \mathsf{int} \rangle\} \\ \ell : \mathsf{loc}\{b : \mathsf{chan}\langle \mathsf{loc}[\mathsf{chan}\langle \mathsf{bool} \rangle] \rangle\} \\ m : \mathsf{loc}\{d : \mathsf{chan}\langle \mathsf{loc}[\mathsf{chan}\langle \mathsf{bool} \rangle] \rangle\} \end{array} \right\}$$

Let P be the following system:

$$\begin{array}{l} k[\![m :: d!\langle k[a] \rangle]\!] \\ \mid m[\![d?(z[x])\ \ell :: b!\langle z[x] \rangle]\!] \\ \mid \ \ell[\![b?(z[x])\ z :: x!\langle \mathsf{t} \rangle]\!] \end{array}$$

Here k communicates the name of its integer channel a to m, using the channel d local to m. Then m misinforms ℓ about the type of a at k: the communication along b fools ℓ into believing that a is a boolean channel. Subsequently ℓ attempts to send an agent to k that violates the type of local resource a, by sending a boolean value where an integer is expected.

The reader can check that according to our runtime semantics the first code movement between k and m is allowed:

$$\Delta \triangleright\ k[\![m :: d!\langle k[a] \rangle]\!] \longmapsto m[\![d!\langle k[a] \rangle]\!]$$

as local type checking of the migrating agent succeeds, $\Delta(m) \vdash d!\langle k[a] \rangle$. The local channel d is used correctly and since the value transmitted, $k[a]$, is non-local it is essentially not examined (only the number of names is checked, not their types).

The local communication at m on channel d now occurs and the second code movement between m and ℓ is also allowed,

$$\Delta \triangleright m[\![d!\langle k[a] \rangle]\!] \mid m[\![d?(z[x])\ \ell :: b!\langle z[x] \rangle]\!] \longmapsto \cdot \longmapsto \ \ell[\![b!\langle k[a] \rangle]\!]$$

because the migrating thread, $b!\langle k[a] \rangle$, is also successful in its type check against local resources, $\Delta(\ell)$. The local communication along b now occurs

$$\Delta \triangleright\ \ell[\![b!\langle k[a] \rangle]\!] \mid \ell[\![b?(z[x])\ z :: x!\langle \mathsf{t} \rangle]\!] \longmapsto \ell[\![k :: a!\langle \mathsf{t} \rangle]\!]$$

However the next potential move, the migration of the thread $a!\langle \mathsf{t} \rangle$ from ℓ to k, is disallowed by the rule $(\mathsf{r\text{-}move})$; the thread is locally type checked against the resources at k, where a is known to be an integer channel, and its potential misuse is discovered.

TABLE 5 Static typing

Values and Threads: As in Table 4

Systems:

$$(\mathsf{t\text{-}rung})\quad \frac{\mathbb{L} \vdash p}{\Gamma,\, \ell{:}\mathbb{L} \vdash \ell[\![p]\!]} \qquad (\mathsf{t\text{-}runb})\quad \frac{\ell \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \ell[\![p]\!]}$$

$$(\mathsf{t\text{-}newcg})\quad \frac{\Gamma,\, \ell{:}(\mathbb{L},\, a{:}\mathrm{A}) \vdash P}{\Gamma,\, \ell{:}\mathbb{L} \vdash (\nu_\ell a{:}\mathrm{A})\, P} \qquad (\mathsf{t\text{-}newcb})\quad \frac{\ell \notin \mathrm{dom}(\Gamma) \quad \Gamma \vdash P}{\Gamma \vdash (\nu_\ell a{:}\mathrm{A})\, P}$$

$$(\mathsf{t\text{-}newlg})\quad \frac{\Gamma,\, \ell{:}\mathbb{L},\, k{:}\mathrm{K} \vdash P}{\Gamma,\, \ell{:}\mathbb{L} \vdash (\nu_\ell k{:}\mathrm{K})\, P} \qquad (\mathsf{t\text{-}newlb})\quad \frac{\ell \notin \mathrm{dom}(\Gamma) \quad \Gamma \vdash P}{\Gamma \vdash (\nu_\ell k{:}\mathrm{K})\, P}$$

$$(\mathsf{t\text{-}str})\quad \frac{\Gamma \vdash P,\, Q}{\Gamma \vdash \mathsf{nil},\, P \,|\, Q}$$

Configurations:

$$(\mathsf{t\text{-}config})\quad \frac{\Gamma \vdash P}{\Gamma \vdash \Delta \triangleright P}\ \forall \ell \in \mathrm{dom}(\Gamma) : \Gamma(\ell) \leq \Delta(k)$$

4 Static typing

In the runtime semantics misuse of local resources can certainly occur, since there is no requirement that values have the object type specified by the transmitting channel. For example the reduction

$$\Delta \triangleright k[\![a!\langle \mathsf{t} \rangle]\!] \mid k[\![a?(x)\, q]\!] \longmapsto k[\![\mathsf{nil}]\!] \mid k[\![q\{{}^{\mathsf{t}}/_{x}\}]\!]$$

is allowed even if the object type of the channel a at k is int, *i.e.* $\Delta(k) \leq \mathsf{loc}\{a{:}\mathsf{chan}\langle \mathsf{int} \rangle\}$.

We do not assume that all sites respect the typing constraints on their channels. For convenience let us call sites that violate the typing constraints *bad* sites, as they do not play according to the rules. A typical example is the site m, described at the end of the previous section; it receives an integer channel a from k but then attempts to use a to send a boolean value. In contrast a *good* site is one where typing constraints are enforced.

In this section we present a static type system that guarantees that:

> *good* sites cannot be harmed by *bad* sites.

That is, local resources at a good site cannot be misused despite the existence

of, and interaction with, bad sites. We prove Subject Reduction and Type Safety theorems for the type system. Intuitively, Subject Reduction can be interpreted as saying that the integrity of good sites is maintained as computation proceeds, while Type Safety demonstrates that local resources at good sites cannot be misused.

The static typing relation for anonymous networks is defined in Table 5. Judgments are of the form

$$\Gamma \vdash P$$

where Γ is a (open) type environment and P a system. The type environment only records the types of good locations; thus $k \in \mathrm{dom}(\Gamma)$ is to be read "k is good" and $m \notin \mathrm{dom}(\Gamma)$ may be read as "m is bad." If $\Gamma \vdash P$, then those agents in P that are located at sites in the domain of Γ are guaranteed to be "well behaved".

For threads and values, the static typing relation is the same as runtime typing relation given in Table 4. The typing of a located agent $\ell[\![p]\!]$ depends on whether ℓ is good, *i.e.* $\ell \in \mathrm{dom}(\Gamma)$. If ℓ is good, then according to $(\mathsf{t\text{-}rung})$, to infer $\Gamma \vdash \ell[\![p]\!]$ we need to establish that the p is well-typed to run at ℓ, $\Gamma(\ell) \vdash p$. But if ℓ is not good, then it doesn't matter whether p is well-typed; thus $(\mathsf{t\text{-}runb})$ says that if $\ell \notin \mathrm{dom}(\Gamma)$, then $\ell[\![p]\!]$ is well-typed for any thread p.

The typing of new resources at the systems level also depends on whether they are introduced at a good site. For example to infer $\Gamma, \ell{:}\mathbb{L} \vdash (\nu_\ell a{:}\mathrm{A})\,P$, where because of the notation we know ℓ is a good site, $(\mathsf{t\text{-}newcg})$ says that we need to establish $\Gamma, \ell{:}(\mathbb{L}, a{:}\mathrm{A}) \vdash P$, *i.e.* that P is well-typed under the assumption that there is a new resource a at site ℓ. This is in contrast to the case when ℓ is not good, $\ell \notin \mathrm{dom}(\Gamma)$, where according to $(\mathsf{t\text{-}newcb})$ it is sufficient to establish that the system P is well-typed relative to the same typing environment. Note that implicit in these two rules is the assumption that new good sites can only be generated by good sites.

Intuitively $\Gamma \vdash P$ means that in P the good sites, those in the domain of Γ use their local resources in accordance with Γ, whereas the behavior of bad sites are unconstrained. As an example consider the configuration $\Delta \triangleright P$ from Section 3.2. If we let Γ be the typing environment defined by:

$$\Gamma = \left\{ \begin{array}{l} k : \mathsf{loc}\{a{:}\mathsf{chan}\langle\mathsf{int}\rangle\} \\ \ell : \mathsf{loc}\{b{:}\mathsf{chan}\langle\mathsf{loc}[\mathsf{chan}\langle\mathsf{bool}\rangle]\rangle\} \end{array} \right\}$$

Then one can easily check that $\Gamma \vdash \Delta \triangleright P$, that is Δ is consistent with Γ and $\Gamma \vdash P$. Intuitively here we are saying that k and ℓ are good sites which use their local resources correctly whereas no guarantee is made about the local behavior at m. Note, however, that if one considers static typing under Δ, which includes m, then Δ does not type P.

The static typing system satisfies several important properties, such as type

TABLE 6 Runtime Error

$$(\textsf{e-comm})\ \ell[\![a!\langle V\rangle p]\!] \mid \ell[\![a?(X{:}\mathrm{T})\, q]\!] \xrightarrow{err\ell} \quad \text{if } V \not\asymp \mathrm{T}$$

$$(\textsf{e-eq})\ \ell[\![\textsf{if } U = V \textsf{ then } p \textsf{ else } q]\!] \xrightarrow{err\ell} \quad \text{if } U \not\asymp V$$

$$(\textsf{e-new})\ \frac{P \xrightarrow{errk}}{(\nu_\ell e{:}\mathrm{E})\, P \xrightarrow{errk\{\ell/e\}}} \qquad (\textsf{e-str})\ \frac{P \xrightarrow{errk}}{P \mid R \xrightarrow{errk}} \qquad \frac{P \equiv Q \quad Q \xrightarrow{errk}}{P \xrightarrow{errk}}$$

specialization and narrowing, which are stated and proved in Appendix A. These properties are used to establish the following result, which states that well-typing at good sites is preserved by reduction.

THEOREM 4.1 (SUBJECT REDUCTION).
If $\Gamma \vdash \Delta \triangleright P$ *and* $\Delta \triangleright P \longmapsto P'$ *then* $\Gamma \vdash \Delta \triangleright P'$.

Proof. See Appendix A. □

We now discuss the extent to which in the type system precludes the misuse of local resources at good sites. This can be formalized using a notion of runtime error. In this paper, we confine our attention to runtime errors based on arity mismatching. Intuitively, an arity mismatch occurs when the value sent on a channel does not match the type that the recipient expects, or when two structurally dissimilar values are compared using the match construct. To formalize this notion, we define a compatibility relation $U \asymp \mathrm{T}$ between (closed) values and types, and a compatibility relation $U \asymp V$ between (closed) values:

$$\begin{array}{rcl} i \asymp \mathsf{int} & \qquad & i \asymp i' \\ \mathsf{bv} \asymp \mathsf{bool} & & \mathsf{bv} \asymp \mathsf{bv}' \\ e \asymp \mathrm{A} & & e \asymp e' \\ e[d_1, .., d_n] \asymp \mathrm{K}[\mathrm{A}_1, .., \mathrm{A}_n] & & e[d_1, .., d_n] \asymp e'[d'_1, .., d'_n] \\ (V_1, .., V_n) \asymp (\mathrm{T}_1, .., \mathrm{T}_n) & & (V_1, .., V_n) \asymp (V'_1, .., V'_n) \\ \text{if } \forall i\colon V_i \asymp \mathrm{T}_i & & \text{if } \forall i\colon V_i \asymp V'_i \end{array}$$

The definition of runtime error is given in Table 6 as a predicate $P \xrightarrow{err\ell}$, indicating that P is capable of a runtime error at location ℓ. Essentially an error occurs at location ℓ if either two incompatible values are compared at ℓ or an attempt is made to communicate a value along a local channel which is incompatible with the type of the channel. The only non-trivial rule is (e-new) which may effect a change in the name of the location where the error occurs. For example it will report an error at ℓ in $(\nu_\ell k{:}\mathrm{K})\, P$ if there is a runtime error in P at location k.

We should point out that this definition of run-time error is considerably weaker than that employed in [7]; in that paper, the notion of run-time error

took into account not only to arity mismatches but also *access violations*.

THEOREM 4.2 (TYPE SAFETY). *If* $\Gamma \vdash P$ *and* $\ell \in \mathrm{dom}(\Delta)$ *then* $P \stackrel{err\,\ell}{\not\longmapsto}$.

Proof. See Appendix A. □

This theorem, together with Subject Reduction, can be interpreted informally as saying that as reductions proceed local resources cannot be locally misused at good sites, even in systems where not all sites which are necessarily well-behaved.

5 Discussion

Here we make three points which demonstrate the limitations of both the runtime strategy and the typing system.

First the static typing is very weak as it is designed only to eliminate *local* misuse of *local* resources. Let

$$\Gamma = \left\{ \begin{array}{l} \ell : \mathsf{loc}\left\{ \begin{array}{l} b{:}\mathsf{chan}\langle \mathsf{loc}\{d{:}\mathsf{chan}\langle \mathsf{bool}\rangle\}\rangle \\ c{:}\mathsf{chan}\langle \mathsf{loc}\{d{:}\mathsf{chan}\langle \mathsf{int}\rangle\}\rangle \end{array} \right\} \\ k : \mathsf{loc}\left\{ d{:}\mathsf{chan}\langle \mathsf{int}\rangle \right\} \end{array} \right\}$$

Then $\Gamma \vdash \ell[\![c?(z[x])\, b!(z[x])]\!]$ although obviously here there is, at least informally, a misuse of local channels. However formally the values exchanged on c and b are *potentially remote* and therefore these values are not considered local. For example, consider a companion site k which has one local channel d of type $\mathsf{chan}\langle \mathsf{int}\rangle$. Then no runtime error occurs in the system

$$\Gamma \triangleright \left(\begin{array}{l} \ell[\![c?(z{:}\mathsf{loc}\{d{:}\mathsf{chan}\langle \mathsf{int}\rangle\})\, b!\langle z\rangle]\!] \\ |\, k[\![\ell :: (c!\langle k\rangle\, b?(z)\, z :: d!\langle \mathsf{t}\rangle)]\!] \end{array} \right)$$

although one could argue that at location ℓ there is a misuse of the channel b. A runtime error is avoided by the dynamic type-checking; after the communications on c and b the potential move

$$\ell[\![k :: d!\langle \mathsf{t}\rangle]\!] \longmapsto k[\![d!\langle \mathsf{t}\rangle]\!]$$

is blocked because the agent attempting to move to k, $d!\langle \mathsf{t}\rangle$ does not type check against the local resources at k.

To strengthen the static type checking sites must be willing to remember not only local resources but also information about other sites. However since not all sites are well-behaved it would be dangerous to rely on arbitrary information.

The second point is that there is an over reliance on dynamic type-checking to avoid misuse of resources. This can be highlighted by considering another desirable property of a runtime semantics for open systems:

> Movement between good sites should always be allowed, even in the presence of badly behaved sites.

It is obvious from the example discussed in Section 3.2 that our run-time semantics does not satisfy this property. If we use the static environment Γ defined on page 14 then k and ℓ are *good* sites. But as we have seen in Section 3.2 the abuse by m eventually prevents a movement from k and ℓ.

Finally the requirement to dynamically type-check *all* incoming threads is very inefficient. However because of the very weak form of type checking employed it is essential. A site cannot even trust itself! For example suppose we revised the reduction rule $(\mathsf{r\text{-}move})$ to read as follows:

$$\Delta \triangleright \ell[\![k :: p]\!] \longmapsto \Delta \triangleright k[\![p]\!] \qquad \text{if } k = \ell \text{ or } \Delta(k) \vdash p$$

Here the site k trusts itself and therefore does not type check the thread p. However this rule is not safe; Subject Reduction fails and runtime errors may be introduced. As an example, consider the following configuration, which uses the typing environment Γ given at the beginning of this section:

$$\Gamma \triangleright \ell[\![c?(z)\, z :: c!\langle \mathrm{t} \rangle]\!] \mid \ell[\![c!\langle \ell \rangle]\!]$$

The configuration can be typed with respect to Γ itself, but after the communication the result is $\Gamma \triangleright \ell[\![c!\langle \mathrm{t} \rangle]\!]$, which fails to type under Γ. Moreover $\Gamma \triangleright \ell[\![c!\langle \mathrm{t} \rangle]\!]$ induces a runtime error due to the potential misuse of channel c.

In a companion paper we address these issues by introducing a notion of *trust* between sites. The dynamic typechecking is strengthened by making a site record information about trusted sites, which includes itself. While typechecking in this system is computationally more expensive, not all incoming threads need be checked; those originating at trusted sites are allowed through unchecked. The new semantics has two additional desirable properties: movement between good sites is always allowed (as discussed above), and a stronger notion of Type Safety can be guaranteed (because the typing relation is stronger).

6 Related Work

In this paper we have outlined a strategy for ensuring that the integrity of well-behaved sites is not compromised by the presence of potentially malicious mobile agents. Moreover we have formalized the correctness of this strategy in terms of Subject Reduction and Type Safety theorems for a *partial* type system.

In this study we used the language Dπ [7], one of a number of distributed versions of the π-calculus [9]. For other variations see [6, 13]. The languages in [4, 5] are thematically similar although based on somewhat different principles. We have taken advantage of a rich type system for Dπ, originally presented in [7], where not only do channels have the types originally proposed in [11] for the π-calculus, but locations have types broadly similar to those of objects. An even richer type system is also proposed in [7] in which types correspond to *capabilities*, as in [5], and an interesting topic for future research would be the

extension of *partial* typing to these richer types.

Our research is related to proposals for *proof-carrying code* outlined in [10]: code consumers, which in our case are locations, demand of code producers, in our case incoming threads, that their code is accompanied by a proof of correctness. This proof is checked by the consumer before the code is allowed to execute. The correctness is expressed in terms of a public safety policy announced by the consumer and the producer must provide code along with a proof that it satisfies this policy. In our case this safety policy is determined by the location type which records the types of the consumer's resources, and proof checking corresponds to type checking the incoming code against this record. Our work is different in that the correctness proof can be reconstructed efficiently, and therefore the producer need not supply an explict proof.

For other examples of related work within this framework see [8, 14]. For example the former contains a number of schemes for typechecking incoming code for access violations to local private resources. However the language is very different from ours, namely a sequential higher-order functional language, and there is no direct formalization of the fact that distributed systems which employ these schemes are well-behaved.

A very different approach to system security is based on the use of cryptography and signatures. For example [1] presents a π-calculus based language which contain cryptographic constructs which ensure the exchange of data between trusted agents, while [3] contains a description of the application of this approach in a practical setting.

A Proofs

A.1 Properties of the static type system

First we prove two important properties of type systems with subtyping: Type Specialization and Weakening.

PROPOSITION A.1 (TYPE SPECIALIZATION).

If $\mathbb{L} \vdash V{:}\mathrm{T}$ *and* $\mathrm{T} \leq \mathrm{S}$ *then* $\mathbb{L} \vdash V{:}\mathrm{S}$.

Proof. By induction on the judgement $\mathbb{L} \vdash V{:}\mathrm{T}$. If $V{:}\mathrm{T}$ takes the form $V{:}\mathrm{H}$ then S must coincide with H, since there is no non-trivial subtyping on channel types or base types. If $V{:}\mathrm{T}$ has the form $w[\widetilde{u}]{:}\mathrm{L}[\widetilde{\mathrm{A}}]$ then the result is trivial, using $(\mathsf{t\text{-}loc})$. Finally, the case for tuples follows by induction. □

PROPOSITION A.2 (WEAKENING).

- *If* $\mathbb{L} \vdash V{:}\mathrm{T}$ *and* $\mathbb{K} \leq \mathbb{L}$ *then* $\mathbb{K} \vdash V{:}\mathrm{T}$
- *If* $\mathbb{L} \vdash p$ *and* $\mathbb{K} \leq \mathbb{L}$ *then* $\mathbb{K} \vdash p$
- *If* $\Gamma, w{:}\mathbb{L} \vdash P$ *and* $\mathbb{K} \leq \mathbb{L}$ *then* $\Gamma, w{:}\mathbb{K} \vdash P$

Proof. In each case the proof is by induction on the type inference. We examine two examples of proof on threads:

- (t-r). Here $\mathbb{L} \vdash u?(X{:}\mathrm{T})\,q$ because $\mathbb{L} \vdash u{:}\mathsf{chan}\langle \mathrm{T}\rangle$ and $\mathbb{L}, X{:}\mathrm{T} \vdash q$. We can apply the first statement in the proposition to the former, to obtain $\mathbb{K} \vdash u{:}\mathsf{chan}\langle \mathrm{T}\rangle$, while induction to the latter gives $\mathbb{K}, X{:}\mathrm{T} \vdash q$. An application of (t-r) now gives the required $\mathbb{K} \vdash u?(X{:}\mathrm{T})\,q$.
- (t-newc). Here $\mathbb{L} \vdash (\nu a{:}\mathrm{A})\,p$ because $\mathbb{L}, a{:}\mathrm{A} \vdash p$. By α-conversion we can choose a so that it does not appear in $\mathbb{K}$ and therefore by induction we have $\mathbb{K}, a{:}\mathrm{A} \vdash p$. Now an application of (t-newc) gives the required $\mathbb{K} \vdash (\nu a{:}\mathrm{A})\,p$.

We present four cases for the proof on systems.

- (t-rung). Here $\Gamma, w{:}\mathbb{L} \vdash m[\![p]\!]$ because $\mathbb{M} \vdash p$, where $\mathbb{M} \stackrel{def}{=} (\Gamma, w{:}\mathbb{L})(m)$. If m and w are different then we also have $\mathbb{M} = (\Gamma, w{:}\mathbb{K})(m)$ and therefore an application of (t-rung) gives the required $\Gamma, w{:}\mathbb{K} \vdash m[\![p]\!]$. On the other hand if m is the same as w then $\mathbb{M} = \mathbb{L}$. So we can apply the second part of the proposition to $\mathbb{M}$, obtaining $\mathbb{K} \vdash p$. Now (t-rung) also gives the required $\Gamma, w{:}\mathbb{K} \vdash m[\![p]\!]$.
- (t-runb). This case is trivial.
- (t-newlg). Here $\Gamma, w{:}\mathbb{L} \vdash (\nu_\ell m{:}\mathrm{M})\,P$ because $\ell \in \mathrm{dom}(\Gamma, w{:}\mathbb{L})$ and $\Gamma, w{:}\mathbb{L}, m{:}\mathrm{M} \vdash P$. Applying induction we obtain $\Gamma, w{:}\mathbb{K}, m{:}\mathrm{M} \vdash P$. Now (t-newlg) can be applied since $\ell \in \mathrm{dom}(\Gamma, w{:}\mathbb{K})$, to obtain the required $\Gamma, w{:}\mathbb{K} \vdash (\nu_\ell m{:}\mathrm{M})\,P$.
- (t-newcb). Here $\Gamma, w{:}\mathbb{L} \vdash (\nu_\ell a{:}\mathrm{A})\,P$ because $\ell \notin \mathrm{dom}(\Gamma, w{:}\mathbb{L})$ and $\Gamma, w{:}\mathbb{L} \vdash P$. However we also have $\ell \notin \mathrm{dom}(\Gamma, w{:}\mathbb{K})$ and therefore (t-newcb) can also be applied to obtain the required $\Gamma, w{:}\mathbb{K} \vdash (\nu_\ell a{:}\mathrm{A})\,P$. □

The following Restriction Lemma states that if $\Gamma \vdash P$ and some identifier u does not occur free in P then P can also be typed in an environment obtained from Γ by removing all occurrences of u. For any identifier u let $\Gamma \setminus u$ denote the result of removing all occurrences of u from Γ. For example $(\Gamma, u{:}\mathbb{L}) \setminus u$ denotes Γ while $\big(\Gamma, w{:}(\mathbb{L}, u{:}\mathrm{A})\big) \setminus u$ is the same as $(\Gamma \setminus u), w{:}\mathbb{L}$.

LEMMA A.3 (RESTRICTION).
- *If* $\mathbb{L}, v{:}\mathrm{H} \vdash U{:}\mathrm{T}$ *and* $v \notin \mathrm{fid}(U)$ *then* $\mathbb{L} \vdash U{:}\mathrm{T}$
- *If* $\mathbb{L}, v{:}\mathrm{H} \vdash p$ *and* $v \notin \mathrm{fid}(p)$ *then* $\mathbb{L} \vdash p$
- *If* $\Gamma \vdash P$ *and* $v \notin \mathrm{fid}(P)$ *then* $\Gamma \setminus v \vdash P$.

Proof. By induction on the proof of the typing judgment. □

The following corollary follows by an easy induction on V.

COROLLARY A.4.
- *If* $\mathbb{L}, V{:}\mathrm{S} \vdash U{:}\mathrm{T}$ *and* $\mathrm{fid}(V)$ disjoint $\mathrm{fid}(U)$ *then* $\mathbb{L} \vdash U{:}\mathrm{T}$

- *If* $\mathbb{L}, V{:}\mathrm{S} \vdash p$ *and* $\mathrm{fid}(V)$ disjoint $\mathrm{fid}(p)$ *then* $\mathbb{L} \vdash p$ □

A.2 Subject Reduction

We first show that typing is preserved by the structural equivalence. The most complicated case is already covered by the Restriction proposition.

LEMMA A.5 (SCOPE EXTRUSION).
If $e \notin \mathrm{fn}(Q)$ *then* $\Gamma \vdash Q \,|\, (\nu_\ell e{:}\mathrm{T})$ *if and only if* $\Gamma \vdash (\nu_\ell e{:}\mathrm{T})\,(Q \,|\, P)$

Proof. There are two cases. If $\ell \notin \mathrm{dom}(\Gamma)$, then we can reason as follows:

$$\begin{aligned} \Gamma \vdash (\nu_\ell e{:}\mathrm{T})\,(Q \,|\, P) &\iff \Gamma \vdash P \text{ and } \Gamma \vdash Q \\ &\iff \Gamma \vdash P \,|\, (\nu_\ell e{:}\mathrm{T})\, Q \end{aligned}$$

In the case that $\ell \in \mathrm{dom}(\Gamma)$, the argument is slightly different depending on whether e is a channel or a location. As an example we consider the former, and we assume Γ has the form $\Delta, \ell{:}\mathbb{L}$.

$$\begin{aligned} \Gamma \vdash (\nu_\ell a{:}\mathrm{A})\,(Q \,|\, P) &\iff \Delta, \ell{:}(\mathbb{L}, a{:}\mathrm{A}) \vdash (Q \,|\, P) \\ &\iff \Delta, \ell{:}(\mathbb{L}, a{:}\mathrm{A}) \vdash Q \text{ and } \Delta, \ell{:}(\mathbb{L}, a{:}\mathrm{A}) \vdash P \\ &\iff \Delta, \ell{:}\mathbb{L} \vdash Q \text{ and } \Delta, \ell{:}(\mathbb{L}, a{:}\mathrm{A}) \vdash P \text{ by Restriction} \\ &\iff \Gamma \vdash Q \text{ and } \Gamma \vdash (\nu_\ell a{:}\mathrm{A})\, P \\ &\iff \Gamma \vdash Q \,|\, (\nu_\ell a{:}\mathrm{A})\, P \end{aligned}$$ □

PROPOSITION A.6. *If* $P \equiv Q$ *then* $\Gamma \vdash P$ *implies* $\Gamma \vdash Q$

Proof. By induction on the proof that $P \equiv Q$. All of the rules and most of the axioms are very easy to handle; the most difficult case is (`s-extr`), which follows by the previous lemma. As an example, we consider the rule (`s-new`): $\ell[\![(\nu e{:}\mathrm{T})\, p]\!] \equiv (\nu_\ell e{:}\mathrm{T})\, \ell[\![p]\!]$.

If $\ell \notin \mathrm{dom}(\Gamma)$ then it is immediate as both sides trivially type with respect to Γ. So suppose $\ell \in \mathrm{dom}(\Gamma)$, and, as an example, that $e{:}\mathrm{T}$ is $a{:}\mathrm{A}$. Let Γ have the form $\Delta, \ell{:}\mathbb{L}$. Then $\Gamma \vdash \ell[\![(\nu a{:}\mathrm{A})\, p]\!]$ if and only if $\mathbb{L}, a{:}\mathrm{A} \vdash p$ while

$$\begin{aligned} \Gamma \vdash (\nu_\ell a{:}\mathrm{A})\, \ell[\![p]\!] &\iff \Delta, \ell{:}(\mathbb{L}, a{:}\mathrm{A}) \vdash \ell[\![p]\!] \\ &\iff \mathbb{L}, a{:}\mathrm{A} \vdash p \end{aligned}$$ □

As is normally the case the proof of Subject Reduction depends on the fact that, in some sense, typing is preserved by substitution. To prove this fact the following lemma will be useful:

LEMMA A.7. *If* $\mathbb{L} \vdash V{:}\mathrm{S}$ *and* $\mathbb{L}, X{:}\mathrm{S} \vdash U{:}\mathrm{T}$ *then* $\mathbb{L} \vdash U\{{}^{V}/_{X}\}{:}\mathrm{T}$.

Proof. By induction on the structure of X.

Suppose X is a variable, *i.e.* $X = x$ for some x. We then proceed by induction on U. If U is a base value or has the form $u[\widetilde{v}]$, then the result is trivial. If U is a tuple we can apply the inner induction hypothesis. If U is x, then $U\{\!\{{}^{V}\!/_{x}\}\!\} = V$ and, by the typing rules, $\mathrm{S} \leq \mathrm{T}$; using type specialization, we can therefore conclude $\mathbb{L} \vdash U\{\!\{{}^{V}\!/_{x}\}\!\}{:}\mathrm{T}$, as required. If U is an identifier u that is different from x then $x \notin \mathrm{fid}(U)$ and $U\{\!\{{}^{V}\!/_{x}\}\!\} = U$; therefore by the Restriction Lemma $\mathbb{L} \vdash U\{\!\{{}^{V}\!/_{x}\}\!\}{:}\mathrm{T}$, as required.

Suppose X has the form $z[\widetilde{x}]$, so $\mathbb{L}, X{:}\mathrm{S} = \mathbb{L}$ and $\mathrm{fid}(X)$ disjoint $\mathrm{dom}(\mathbb{L})$ and therefore $U\{\!\{{}^{V}\!/_{X}\}\!\} = U$. By Corollary A.4, $\mathbb{L} \vdash U{:}\mathrm{T}$.

If X is a tuple, the result follows by induction. □

PROPOSITION A.8 (SUBSTITUTION). *If* $\mathbb{L} \vdash V{:}\mathrm{T}$ *and* $\mathbb{L}, X{:}\mathrm{T} \vdash p$ *then* $\mathbb{L} \vdash p\{\!\{{}^{V}\!/_{X}\}\!\}$.

Proof. For convenience we use p' to denote $p\{\!\{{}^{V}\!/_{X}\}\!\}$ (and employ similar notation for other syntactic categories).

As before, the proof is by induction on the structure of X. The cases for tuples $\widetilde{Y}$ and structured values $z[\widetilde{x}]$ are as before. We present the case where $X = x$, for some variable x. In this case we proceed by a second induction on the inference $\mathbb{L}, x{:}\mathrm{T} \vdash p$. We present the cases that involve binders.

- (t-r). Here $\mathbb{L}, x{:}\mathrm{T} \vdash u?(Y{:}\mathrm{S})\, q$ because $\mathbb{L}, x{:}\mathrm{T} \vdash u{:}\mathsf{chan}\langle \mathrm{T} \rangle$ and $\mathbb{L}, x{:}\mathrm{T}, Y{:}\mathrm{S} \vdash q$. Applying the previous lemma to $\mathbb{L}, x{:}\mathrm{T} \vdash u{:}\mathsf{chan}\langle \mathrm{T} \rangle$, we obtain $\mathbb{L} \vdash u'{:}\mathsf{chan}\langle \mathrm{T} \rangle$. In addition, $\mathbb{L}, x{:}\mathrm{T}, Y{:}\mathrm{S} \vdash q$ can be rewritten as $\mathbb{L}, Y{:}\mathrm{S}, x{:}\mathrm{T} \vdash q$. By the internal induction, we obtain $\mathbb{L}, Y{:}\mathrm{S} \vdash q'$. Now (t-r) gives the required $\mathbb{L} \vdash u'?(Y{:}\mathrm{S})\, q'$.
- (t-newc). Here $\mathbb{L}, x{:}\mathrm{T} \vdash (\nu a{:}\mathrm{A})\, p$ because $\mathbb{L}, x{:}\mathrm{T}, a{:}\mathrm{A} \vdash p$. The environment can be rewritten as $\mathbb{L}, a{:}\mathrm{A}, x{:}\mathrm{T}$, and therefore induction can be applied to obtain $\mathbb{L}, a{:} \vdash p'$. The result follows by an application of (t-newc). □

THEOREM (4.1, SUBJECT REDUCTION).
If $\Gamma \vdash \Delta \triangleright P$ *and* $\Delta \triangleright P \longmapsto P'$ *then* $\Gamma \vdash \Delta \triangleright P'$.

Proof. From the hypothesis $\Gamma \vdash \Delta \triangleright P$ we know that $\Gamma \vdash P$ and $\Gamma(\ell) \leq \Delta(\ell)$ for every ℓ in $\mathrm{dom}(\Gamma)$. So it is sufficient to show $\Gamma \vdash P'$, which we do by induction on the derivation of $\Delta \triangleright P \longmapsto P'$. We present a number of representative cases.

- (r-move). Here we have $\Delta \triangleright \ell[\![k :: p]\!] \longrightarrow k[\![p]\!]$ and $\Delta(k) \vdash p$. If $k \notin \mathrm{dom}(\Gamma)$ then the required $\Gamma \vdash k[\![p]\!]$ follows trivially from (t-runb). If $k \in \mathrm{dom}(\Gamma)$ then we know that $\Gamma(k) \leq \Delta(k)$ and therefore, by Weakening, $\Gamma(k) \vdash p$. Now (t-rung) can be applied to obtain $\Gamma \vdash k[\![p]\!]$.
- (r-comm). Here we have

$$\Delta \triangleright \ell[\![a!\langle V \rangle\, p]\!] \mid \ell[\![a?(X{:}\mathrm{T})\, q]\!] \longrightarrow \ell[\![p]\!] \mid \ell[\![q\{\!\{{}^{V}\!/_{X}\}\!\}]\!]$$

If $\ell \notin \mathrm{dom}(\Gamma)$ then the result is trivial from (t-runb). Otherwise $\ell \in \mathrm{dom}(\Gamma)$. From $\Gamma \vdash \ell[\![a!\langle V\rangle\, p]\!] \mid \ell[\![a?(X)\, q]\!]$ we know $\Gamma \vdash \ell[\![a!\langle V\rangle\, p]\!]$ and therefore $\Gamma(\ell) \vdash p$. It follows that $\Gamma \vdash \ell[\![p]\!]$.

It remains to show that $\Gamma \vdash \ell[\![q\{^{V}\!/_{X}\}]\!]$, that is $\Gamma(\ell) \vdash q\{^{V}\!/_{X}\}$. Again from the hypothesis we know $\Gamma \vdash \ell[\![a?(X{:}\mathrm{T})\, q]\!]$ from which we can conclude that $\Gamma(\ell) \vdash u{:}\mathsf{chan}\langle \mathrm{T}\rangle$ and $\mathbb{L}, X{:}\mathrm{T} \vdash q$. From $\Gamma \vdash \ell[\![a!\langle V\rangle\, p]\!]$ we know that $\Gamma(\ell) \vdash V{:}\mathrm{S}$ for some S for which we also have $\Gamma(\ell) \vdash u{:}\mathsf{chan}\langle \mathrm{S}\rangle$. In our typing system this must mean that S and T coincide. We may therefore apply the Substitution lemma to obtain the required $\Gamma(\ell) \vdash q\{^{V}\!/_{X}\}$.

(r-new). We consider the case:

$$\Delta, \ell{:}\mathrm{L} \triangleright (\nu_\ell a{:}\mathrm{A})\, P \longrightarrow (\nu_\ell a{:}\mathrm{A})\, P' \quad \text{because} \quad \Delta, \ell{:}(\mathrm{L}, a{:}\mathrm{A}) \triangleright P \longrightarrow P'$$

First suppose $\ell \in \mathrm{dom}(\Gamma)$. Since $\Gamma \vdash \Delta, \ell{:}\mathrm{L} \triangleright (\nu_\ell a{:}\mathrm{A})\, P$ we know Γ can be written as $\Gamma', \ell{:}\mathrm{L}'$, where $\mathrm{L}' \leq \mathrm{L}$ and therefore $\Gamma', \ell{:}(\mathrm{L}', a{:}\mathrm{A}) \vdash P$. We can now apply induction to obtain $\Gamma', \ell{:}(\mathrm{L}', a{:}\mathrm{A}) \vdash P'$, to which (t-newcg) can be applied to obtain the required $\Gamma \vdash (\nu_\ell a{:}\mathrm{A})\, P'$.

If $\ell \notin \mathrm{dom}(\Gamma)$ then by (t-newcb) it is sufficient to prove $\Gamma \vdash P'$. In this case $\Gamma \vdash \Delta, \ell{:}\mathrm{L} \triangleright (\nu_\ell a{:}\mathrm{A})\, P$ yields $\Gamma \vdash \Delta, \ell{:}(\mathrm{L}, a{:}\mathrm{A}) \triangleright P$ to which induction can be applied to give the required $\Gamma \vdash P'$.

(r-str). This case follows using induction and Proposition A.6. □

A.3 Type Safety

We first show that the typing system is "compatible" with the compatibility relation $\asymp$.

LEMMA A.9.
- $\mathbb{L} \vdash V{:}\mathrm{T}$ *implies* $V \asymp \mathrm{T}$
- $\mathbb{L} \vdash V{:}\mathrm{T}$ *and* $\mathbb{L} \vdash U{:}\mathrm{T}$ *implies* $V \asymp U$

Proof. A straightforward inductive argument, in the first case on the derivation of $\mathbb{L} \vdash V{:}\mathrm{T}$ and in the second on the structure of the type T. □

THEOREM (4.2, TYPE SAFETY). *If* $\Gamma \vdash P$ *and* $\ell \in \mathrm{dom}(\Gamma)$ *then* $P \overset{err\,\ell}{\nrightarrow}$.

Proof. By induction on the proof that $P \overset{err\,\ell}{\longmapsto}$, we show that if $\ell \in \mathrm{dom}(\Gamma)$ and $P \overset{err\,\ell}{\longmapsto}$ then $\Gamma \not\vdash P$, which is sufficient to establish the theorem. Let $\mathbb{L}$ denote $\Gamma(\ell)$.

(e-comm). In this case we have $\ell[\![a!\langle V\rangle\, p]\!] \mid \ell[\![a?(X{:}\mathrm{T})\, q]\!] \overset{err\,\ell}{\longmapsto}$ because $V \not\asymp \mathrm{T}$. Now suppose, for a contradiction, that $\Gamma \vdash \ell[\![a!\langle V\rangle\, p]\!] \mid \ell[\![a?(X{:}\mathrm{T})\, q]\!]$. Because there is no subtyping on channel types we know $\mathbb{L} \vdash \mathsf{chan}\langle \mathrm{T}\rangle$ and $\mathbb{L} \vdash \mathsf{chan}\langle \mathrm{S}\rangle$ implies $\mathrm{T} = \mathrm{S}$. From the alleged typing we can therefore conclude that $\mathbb{L} \vdash V{:}\mathrm{T}$. By the first part of Lemma A.9, we have $V \asymp \mathrm{T}$ which contradictions $V \not\asymp \mathrm{T}$.

(e-eq). Here we have $\ell[\![\mathsf{if}\ U = V\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] \stackrel{err\,\ell}{\longmapsto}$ because $U \not\asymp V$. If we assume $\Gamma \vdash \ell[\![\mathsf{if}\ U = V\ \mathsf{then}\ p\ \mathsf{else}\ q]\!]$ then we must have $\Gamma \vdash U$:T and $\Gamma \vdash V$:T for some T. Now applying the second part of Lemma A.9 we obtain a contradiction to $U \not\asymp V$.

(e-new). First suppose that $(\nu_k e{:}\mathrm{E})\,P \stackrel{err\,\ell}{\longmapsto}$ because $P \stackrel{err\,\ell}{\longmapsto}$ and $e \neq \ell$. If $k \notin \mathrm{dom}(\Gamma)$ then by induction we have $\Gamma \not\vdash P$ and by either (t-newcb) or (t-newlb) we can conclude $\Gamma \not\vdash (\nu_k e{:}\mathrm{E})\,P$.

So suppose $k \in \mathrm{dom}(\Gamma)$. As an example suppose e:E is a location; the case when it is a channel is similar. Applying induction we have $\Gamma, e{:}\mathrm{E} \not\vdash P$ and therefore by (t-newlg) we can conclude $\Gamma \not\vdash (\nu_k e{:}\mathrm{E})\,P$.

Finally suppose $(\nu_\ell k{:}\mathrm{E})\,P \stackrel{err\,\ell}{\longmapsto}$ because $P \stackrel{err\,k}{\longmapsto}$. By α-conversion we can assume that k does not appear in Γ and so we can apply induction to obtain $\Gamma, k{:}\mathrm{E} \not\vdash P$. We know that $\ell \in \mathrm{dom}(\Gamma)$ and therefore by (t-newlg) we can conclude $\Gamma \not\vdash (\nu_\ell k{:}\mathrm{E})\,P$.

(e-str). Straightforward, using induction and Proposition A.6. □

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report 414, University of Cambridge Computer Laboratory, January 1997.

[2] *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[3] B. Bershad, B. Savage, P. Pardyak, E. Sirer, D. Becker, M. Fiuczynski, C. Chambers, and S. Eggers. Extensibility, safety and performance in the SPIN operating system. In *Symposium on Operating Systems Principles*, pages 267–284, 1997.

[4] L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computational Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Berlin: Springer-Verlag, 1998.

[5] R. DeNicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. Technical report, Dipartimento di Sistemi e Informatica, Università di Firenze, 1997.

[6] C. Fournet, G. Gonthier, J.J. Levy, L. Marganget, and D. Remy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *CONCUR: Proceedings of the International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, August 1996. Berlin: Springer-Verlag.

[7] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. Technical Report 2/98, University of Sussex, Computer Science, 1998.

[8] Xavier Leroy and Francois Rouaix. Security properties of typed applets. In ACM-POPL [2].

[9] Robin Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, October 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[10] George Necula. Proof-carrying code. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, Paris,France, January 1997. ACM Press.

[11] Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. Extended abstract in LICS '93.

[12] James Riely and Matthew Hennessy. A typed language for distributed mobile processes. In ACM-POPL [2].

[13] Peter Sewell. Global/local subtyping for a distributed π-calculus. Technical Report 435, Computer Laboratory, University of Cambridge, August 1997.

[14] R. Stata and M. Abadi. A type system for java bytecode subroutines. In ACM-POPL [2].