Trust and Partial Typing in Open Systems of Mobile Agents

James Riely (jriely@cs.depaul.edu)*
DePaul,CTI

Matthew Hennessy (matthewh@cogs.sussex.ac.uk)†
University of Sussex

March 3 2003

Abstract. We present a *partially-typed* semantics for Dπ, a distributed π-calculus. The semantics is designed for mobile agents in *open* distributed systems in which some sites may harbor malicious intentions. Nonetheless, the semantics guarantees traditional type-safety properties at *good* locations by using a mixture of static and dynamic type-checking. We show how the semantics can be extended to allow *trust* between sites, improving performance and expressiveness without compromising type-safety.

1. Introduction

In [13] we presented a type system for controlling the use of resources in a distributed system, or network. The type system guarantees two properties:

- resource access is always *safe*, for example, integer resources are always accessed with integers and string resources are always accessed with strings, and
- resource access is always *authorised*, that is, resources may only be accessed by agents that have been granted *permission* to do so.

While these properties are desirable, they are properties of a network *as a whole*. In *open systems* it is impossible to verify the system as a whole, for example, to "type-check the web". In this paper, we present type systems and semantics for open systems that guarantee the first property above.

Any treatment of open systems must assume some underlying security mechanisms for communication between *sites*, or *locations*. One approach would be to add security features directly in the language, as in Abadi and Gordon's Spi calculus [1]. In such languages code signatures and nonces are directly manipulable as program objects. Here we take a more abstract approach, presenting a "secure" semantics

* Research funded by NSF grant EIA-9805604
† Research funded by EPSRC project GR/K60701

for a language without explicit security features. Of the underlying communication mechanism, we assume only that it delivers packets uncorrupted and that the source of a packet can be reliably determined. In wide-area networks, a low-level signature mechanism may be required to realise these assumptions.

We start our development from the following principles:

1. Sites are divided into two groups: the *good*, or typed, and the *bad*, or untyped, the latter of which may harbour malicious agents.
2. Malicious agents should not be able to corrupt computation at good sites; however, not all agents at bad sites are necessarily malicious. Thus, the static notions of good and bad should not be used to prevent actions by an agent; rather, some form of dynamic typechecking is necessary.
3. Because agent interaction is commonplace, agent movement, rather than interaction, should be subject to dynamic typechecking.

In practice, the distinction between good and bad sites is made relative to a particular administrative domain. In the narrowest setting, only one particular virtual machine (VM), or location, might be considered good, or well-typed, whereas all other machines on the network are considered potentially malicious. In this case, the goal of a security mechanism is to protect the local machine from misuse, while at the same time allowing certain code from other machines to be installed locally. More generally, the distinction between good and bad might be drawn between intra- and inter-net, with corporate or departmental machines protected by well-typing.

Here we are interested in preventing misuse based on type-mismatching — for example, a foreign agent attempting to access an area of memory which is unallocated, or is allocated to a different VM; or an agent attempting to read an integer location as an array, and thus gaining access to arbitrarily large areas of memory. Such type violations may lead to core dumps, information leakage or the spread of viruses and other virtual pestilence.

We study these issues in the formal setting of Dπ [13], a distributed variant of the π-calculus [16]. In Dπ resources reside at locations and mobile agents may move from site to site, interacting via local resources to affect computations. The typing system of Dπ is based on *location types* which describe the resources available at a site. For example

$$\mathsf{loc}\{puti{:}\mathsf{res}\langle\mathsf{int}\rangle,\ geti{:}\mathsf{res}\langle\mathsf{int}\rangle,\ putl{:}\mathsf{res}\langle\mathsf{loc}\rangle,\ getl{:}\mathsf{res}\langle\mathsf{loc}\rangle\}$$

is the type of a location with four resources, two for manipulating integers and two for manipulating location names; as with many calculi based on the π-calculus, resources in Dπ are modelled as *communication channels*. A feature which distinguishes Dπ from related languages [10, 2, 24] however is that resource names have only local significance, that is, resource names are unique locally, but not globally across the network. This property reflects the open-ended nature of open systems such as the internet.

To formalise the notion of "bad sites" in Dπ, we add a new location type, lbad, to the language. Agents residing at locations of type lbad are effectively untyped, as are references to resources at bad locations, regardless of where these references occur. This weaker form of typing is achieved by adding a new inference rule to the typing system and a new form of subtyping. We call the resulting typing system a *partial* typing system, as agents and resources at bad sites are untyped. Nevertheless partial typing ensures that resources at *good* sites are *not* misused.

The weakness of partial typing allows for the existence of malicious agents at bad sites. Further, since agents can move, unprotected good sites can easily be corrupted; an example of this phenomenon is described in Section3.2. Technically this means that partial typing is not preserved by the standard reduction semantics of Dπ; a good site may cease to be well-typed after a migration from an untyped site. The objective of this paper is the formalisation of a protection policy for good sites against such malicious attacks.

As in [25, 19, 14, 18], the basic idea is to require that code be verified before it is loaded locally. Unlike these references, however, our work is explicitly agent-based, and allows incoming agents to carry references to resources distributed throughout the network; further, our approach supports the introduction of *trust* between sites, as described below.

Verification of incoming agents takes the form of dynamic typechecking, where incoming code is compared against a *filter* for the target site. Filters provide an incomplete, or partial, view of the types of the resources in the network, both local and remote. Since the information in filters is incomplete, the dynamic typechecking algorithm must be able to certify agents even when the filter contains little or no information about the agent's site of origin; otherwise, it would forbid too many migrations. This is potentially very dangerous as malicious agents may lie about resources at their origin or at a third-party site.

We avoid this danger by developing an adequate semantics based on the notion of *authority*. An agent moving from location k to ℓ is dynamically typechecked under the authority of k, using the filter for ℓ; every resource access must be verified either by the filter or the authority; in general the willingness of ℓ to use code or third-party information emanating from a source k will depend on its belief in k. One could envisage a more general concept of *authority for code* but in this paper we confine our attention to a simple form, based on it's source. The full development is given in Section 4, where we prove Subject Reduction and Type Safety theorems for this semantics, ensuring that resource access at good locations is always type-safe. This approach should be contrasted with that of [12] (outlined in Section B), which gives an adequate semantics for networks in which the authority of incoming agents cannot reliably be determined.

One drawback of this framework is that every agent must be dynamically typechecked when moving from one site to another. To alleviate this burden, in Section 5 we introduce a relationship of *trust* between locations, formalised using the location type ltrust. We then modify the operational semantics so that agents originating at trusted locations need not be typechecked. Although technically this is a simple addition to the type system, it is also very expressive. The result is that the network is divided into *webs of trust* and agents can only gain entry to a web of trust via typechecking. Once entry to a web of trust has been earned, however, an agent can move freely around the web; it will only be typechecked again if it leaves a web and subsequently wishes to reenter. Moreover these webs of trust may grow dynamically as incoming agents inform sites of other sites that they can trust.

We now present a quick overview of the remainder of the paper. In Section 2 we first review Dπ and its *standard* semantics, including the standard static typing system. Section 3 introduces the notion of partial typing and shows that partial typing is not preserved by the standard reduction relation. The next two sections contain the main contributions of the paper. Section 4 presents the formalisation of filters and dynamic typing, showing how these are incorporated into the runtime semantics. In Section 5 this framework is extended to include *trust*. Both sections include several examples, as well as proofs of Subject Reduction and Type Safety. In Section 6 we discuss the design of the semantics and describe some of its limitations, pointing to topics for further research. The paper ends with a brief survey of related work.

2. The Language and Standard Typing

In this section we review the syntax and standard semantics of Dπ. For a full treatment of the language, including many examples, see [13]. Our formalisation of the language differs slightly from that of [13], as discussed in the conclusion.

Table I. Syntax of names e, values u, patterns X, threads P, and networks M.

e	$::=$	k, ℓ, m	Location	u, v, w	$::=$	bv	Base Value
	$\mid$	a, b, c	Resource		$\mid$	e	Name
X, Y	$::=$	x, y, z	Variable		$\mid$	x, y, z	Variable
	$\mid$	$(X_1, .., X_n)$	Tuple		$\mid$	$(u_1, .., u_n)$	Tuple
	$\mid$	$(X_1, .., X_n)@Y$	Located		$\mid$	$(u_1, .., u_n)@w$	Located Value
P, Q, R	$::=$	stop	Termination	M, N	$::=$	$\mathbf{0}$	Empty
	$\mid$	$P \mid Q$	Composition		$\mid$	$M \mid N$	Composition
	$\mid$	$(\nu e{:}\mathrm{T})\, P$	Restriction		$\mid$	$(\nu_k\, e{:}\mathrm{T})\, N$	Restriction
	$\mid$	$\mathsf{goto}\, u.\, P$	Movement		$\mid$	$k[\![P]\!]$	Agent
	$\mid$	$u!\langle v\rangle\, P$	Output				
	$\mid$	$u?(X{:}\mathrm{T})\, P$	Input				
	$\mid$	$*P$	Replication				
	$\mid$	$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Matching				

2.1. Syntax

The syntax is given in Table I, although discussion of types, T, is postponed to Section 2.3. The syntax is parameterised with respect to the following syntactic sets, which we assume to be disjoint:

- *Base*, of *base values*, ranged over by bv,
- *Loc*, of *location names*, ranged over by k, ℓ, m,
- *Res*, of *resource names*, ranged over by a, b, c,
- *Var*, of *variables*, ranged over by x, y, z.

Names, e, include location names and resource names. *Values*, u–w, include base values, names, variables and tuples of values. We also have *located values* of the form $(u_1, .., u_n)@w$, used for the transmission of *non-local information*. Intuitively this represents a tuple of resource names, $(u_1, .., u_n)$, located at the site w; the need for such values is discussed at length in [13].

To avoid too many separate categories of meta-variables we occasionally use u, v, w to range over restricted classes of values, such as $\mathit{Var} \cup \mathit{Loc}$ or $\mathit{Var} \cup \mathit{Res}$. Such cases should be clear from context; moreover the typing system, when given in Section 2.3, will enforce reasonable assumptions. *Patterns*, X, Y, are designed so that arbitrary values can be decomposed into their constituents; we require that they be linear, that is, that each variable appear at most once.

The main syntactic categories of the language are as follows:

- *Threads*, P–R, are terms of the ordinary polyadic π-calculus [15] with additional constructs for movement and restriction of locations.

- *Agents*, $k[\![P]\!]$, are located threads.
- *Networks*, M–N, are collections of agents combined using the static combinators of composition and restriction.

As an example of a network, consider the term:

$$\ell[\![P]\!] \mid (\nu_\ell a{:}\mathrm{T})\,(\ell[\![Q]\!] \mid k[\![R]\!])$$

This network 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 knowledge of a private resource a of type T, allocated at ℓ and unknown to P.

Notation 1. *We adopt several notational conventions, as in [13].*

- *In the concrete syntax,* $\mathsf{goto}\,u$ *has greater binding power than composition. Thus '*$\mathsf{goto}\,k.\,P \mid Q$*' should be read '*$(\mathsf{goto}\,k.\,P) \mid Q$*'. We adopt several standard abbreviations. For example, we routinely drop type annotations when they are not of interest. We omit trailing occurrences of* stop *and often denote tuples and other groups using a tilde. For example, we write* $\widetilde{u}$ *instead of* $u_1, .., u_n$ *and* $\widetilde{u}{:}\widetilde{\mathrm{T}}$ *instead of* $u_1{:}\mathrm{T}_1, .., u_n{:}\mathrm{T}_n$*. We also write '*$\mathsf{if}\ u = v\ \mathsf{then}\ P$*' instead of '*$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else\ stop}$*' and '*$\mathsf{if}\ u \neq v\ \mathsf{then}\ Q$*' instead of '*$\mathsf{if}\ u = v\ \mathsf{then\ stop\ else}\ Q$*.'*
- *We assume the standard notion of* free *and* bound *occurrences of variables and names in networks and threads. The variables in the pattern* X *are bound by the input construct* $u?(X)\,P$*, the scope is* P*. The name* e *is bound by the restrictions* $(\nu e)\,P$ *and* $(\nu_k e)\,N$*, the scopes are* P *and* N*, respectively; note however that* k *is considered to be free in the latter. A term with no free variables is* closed*. The functions* $\mathrm{fn}(P)$ *and* $\mathrm{fv}(P)$ *return respectively the sets of free names and free variables occurring in* P*.*
- *We also assume a standard notion of* substitution*, where* $P\{\!\!\{{}^{u}\!/\!{}_{x}\}\!\!\}$ *denotes the capture-avoiding substitution of* u *for* x *in* P*. The notation* $P\{\!\!\{{}^{V}\!/\!{}_{X}\}\!\!\}$ *generalises this in an obvious way as a sequence of substitutions, following the structure of the pattern* X*.*
- *In the sequel we identify terms up to renaming of bound names and variables.*

Table II. Standard Reduction

Structural congruence:

$$\begin{array}{rl}
\textsf{(s-extr)} & M \mid (\nu_k\, e{:}\mathrm{T})\, N \equiv (\nu_k\, e{:}\mathrm{T})\,(M \mid N) \quad \text{if } e \notin \mathrm{fn}(M) \\
\textsf{(s-garbage}_1\textsf{)} & (\nu_k\, e{:}\mathrm{T})\ \mathbf{0} \equiv \mathbf{0} \\
\textsf{(s-garbage}_2\textsf{)} & k[\![\mathsf{stop}]\!] \equiv \mathbf{0} \\
\textsf{(s-copy)} & k[\![{*P}]\!] \equiv k[\![P]\!] \mid k[\![{*P}]\!]
\end{array}$$

Reduction precongruence:

$$\begin{array}{rl}
\textsf{(r-move)} & k[\![\mathsf{goto}\, \ell.\, P]\!] \longmapsto \ell[\![P]\!] \\
\textsf{(r-new)} & k[\![(\nu\, e{:}\mathrm{T})\, P]\!] \longmapsto (\nu_k\, e{:}\mathrm{T})\, k[\![P]\!] \quad \text{if } e \neq k \\
\textsf{(r-split)} & k[\![P \mid Q]\!] \longmapsto k[\![P]\!] \mid k[\![Q]\!] \\
\textsf{(r-comm)} & k[\![a!\langle v\rangle\, P]\!] \mid k[\![a?(X)\, Q]\!] \longmapsto k[\![P]\!] \mid k[\![Q\{\!\!|{}^{v}\!/\!{}_{X}|\!\!\}]\!] \\
\textsf{(r-eq}_1\textsf{)} & k[\![\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \longmapsto k[\![P]\!] \\
\textsf{(r-eq}_2\textsf{)} & k[\![\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \longmapsto k[\![Q]\!] \quad \text{if } u \neq v
\end{array}$$

2.2. Standard Reduction

The standard reduction semantics is given in Table II. The structural congruence ($M \equiv N$) and reduction precongruence ($M \longmapsto M'$) both relate *closed* network terms. The main reduction relation we are interested in is $(\longrightarrow) = (\equiv \cdot \longmapsto \cdot \equiv)$.

The structural congruence is defined to be the least congruence relation[1] on networks that satisfies the commutative monoid laws for composition[2] and the axioms given in Table II. The axioms provide means for the extension of the scope of a name, for garbage collection of unused names and terminated threads, and for the replication of agents.

The reduction relation $\longmapsto$ is defined to be the least precongruence relation on networks which satisfies the reduction axioms of Table II. The axioms for communication and matching are taken directly from the π-calculus, with a few changes to accommodate the fact that agents are explicitly located. Note that communication can only occur between co-located agents.

The most important new rule is (r-move), $k[\![\mathsf{goto}\, \ell.\, P]\!] \longmapsto \ell[\![P]\!]$, which states that an agent located at k can move to ℓ using the move operator $\mathsf{goto}\, \ell.\, P$. Also significant is (r-new), $k[\![(\nu e{:}\mathrm{T})\, P]\!] \longmapsto (\nu_k\, e{:}\mathrm{T})\, k[\![P]\!]$, which states that a name created by a thread can become available across the network. Note that when a new name is lifted out of an

[1] A relation $\prec$ is a precongruence on networks if $N \prec N'$ implies $N \mid M \prec N' \mid M$, $M \mid N \prec M \mid N'$, and $(\nu_k\, e{:}\mathrm{T})\, N \prec (\nu_k\, e{:}\mathrm{T})\, N'$. A relation is a congruence if it is both an equivalence and a precongruence.

[2] The monoid laws are: $M \mid \mathbf{0} \equiv M$, $M \mid N \equiv N \mid M$, and $M \mid (N \mid O) \equiv (M \mid N) \mid O$.

agent, the network-level restriction records the name of the location which allocated the name; these location tags are used only for static typing. Finally, the rule $(\mathsf{r\text{-}split})$, $k[\![P \mid Q]\!] \longmapsto k[\![P]\!] \mid k[\![Q]\!]$, allows an agent to spawn off sub-agents which are able to move around the network independently. The only reduction rules that vary significantly in later sections are $(\mathsf{r\text{-}move})$ and $(\mathsf{r\text{-}new})$.

Example 1. *As an example, suppose that we wish to write a network with two agents, one a* server *based at ℓ and the other a* client *based at k. The server could take the form*

$$\ell[\![b?(x@z)\ Q]\!]$$

where some remote channel is awaited at b; the name of the channel is to be bound to x, and its location to z. A typical example of the serving code Q that uses the incoming structured value $x@z$ could be $\mathsf{goto}\, z.\, x!\langle 1\rangle$*. The client at k could take the form*

$$k[\![(\nu a)\ (a?(y)\, P \mid \mathsf{goto}\, \ell.\, b!\langle a@k\rangle)]\!]$$

which generates a new reply channel a, located at k, sends it to the server at b and awaits a response at the newly generated a. This network would reduce as follows:

$$\begin{array}{lr}
\ell[\![b?(x@z)\ Q]\!] \mid k[\![(\nu a)\ (a?(y)\, P \mid \mathsf{goto}\, \ell.\, b!\langle a@k\rangle)]\!] & \\
\longrightarrow \ell[\![b?(x@z)\ Q]\!] \mid (\nu_k a)\, (k[\![a?(y)\, P \mid \mathsf{goto}\, \ell.\, b!\langle a@k\rangle]\!]) & (\mathsf{r\text{-}new}) \\
\longrightarrow \ell[\![b?(x@z)\ Q]\!] \mid (\nu_k a)\, (k[\![a?(y)\, P]\!] \mid k[\![\mathsf{goto}\, \ell.\, b!\langle a@k\rangle]\!]) & (\mathsf{r\text{-}split}) \\
\longrightarrow \ell[\![b?(x@z)\ Q]\!] \mid (\nu_k a)\, (k[\![a?(y)\, P]\!] \mid \ell[\![b!\langle a@k\rangle]\!]) & (\mathsf{r\text{-}move}) \\
\longrightarrow (\nu_k a)\, (\ell[\![Q\{\!|{}^{a@k}\!/\!_{x@z}|\!\}]\!] \mid k[\![a?(y)\, P]\!]) & (\mathsf{s\text{-}extr}), (\mathsf{r\text{-}comm}), (\mathsf{s\text{-}garbage_2})
\end{array}$$

Beside each reduction, we have written the axioms used to infer it, omitting mention of the monoid laws.

At this point the server processes the request, with knowledge of the return channel and its location, a at k, and the client is awaiting a response on a.

For more detailed discussion of the reduction semantics the reader is referred to [13]. □

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 language from [13, §5], extended with base types. However all of the results in this paper extend smoothly to the more powerful type system of [13,

§6], which includes resource capabilities and non-trivial subtyping on resource types.

We use uppercase Roman letters to range over types, whose syntax follows:

$$\begin{array}{lll} \text{Resources:} & \mathrm{A}\text{–}\mathrm{D} & ::= \mathsf{res}\langle \mathrm{T}\rangle \\ \text{Locations:} & \mathrm{K}, \mathrm{L} & ::= \mathsf{loc}\{a_1{:}\mathrm{A}_1,\ ..,\ a_n{:}\mathrm{A}_n, x_1{:}\mathrm{B}_1,\ ..,\ x_n{:}\mathrm{B}_n\} \\ \text{Values:} & \mathrm{S}, \mathrm{T} & ::= \mathsf{BT} \mid \mathrm{K} \mid \mathrm{A} \mid (\mathrm{A}_1,\ ..,\ \mathrm{A}_n)@\mathrm{K} \mid (\mathrm{T}_1,\ ..,\ \mathrm{T}_n) \end{array}$$

The syntax provides types for base values, locations, local resources and tuples. Location types are essentially the same as standard record types; to be well-formed each resource name and variable in a location type can appear at most once. As with record types we identify location types up to reordering of their "fields". Thus $\mathsf{loc}\{a{:}\mathrm{A},\ b{:}\mathrm{B}\} = \mathsf{loc}\{b{:}\mathrm{B},\ a{:}\mathrm{A}\}$. We write '$\mathsf{loc}$' for '$\mathsf{loc}\{\,\}$'.

Types of the form $(\widetilde{\mathrm{A}})@\mathrm{K}$ are *dependent* tuple types, which allow communication of non-local resources; values of this type are of the form $(a_1,\ ..,\ a_n)@k$ and we have already seen their use in Example 1. There, with type annotations on the bound variables, the server would take the form

$$\ell[\![b?(x@z{:}\mathrm{A}@\mathsf{loc})\ Q]\!]$$

where A determines the type of the remote channel to be received. Again we refer the reader to [13] for more details.

We require that each resource name and variable in a location type appear at most once.

The subtyping preorder ($\mathrm{T} <: \mathrm{S}$) is discussed at length in [13]. On base types and channel types there is no nontrivial subtyping; for example, $\mathsf{res}\langle \mathrm{T}\rangle <: \mathsf{res}\langle \mathrm{T}'\rangle$ if and only if $\mathrm{T} = \mathrm{T}'$. On location types, the subtyping relation is similar to that traditionally defined for record or object types (although here it is invariant):

$$\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{A}},\ \widetilde{v}{:}\widetilde{\mathrm{B}}\} <: \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{A}}\}$$

On tuples, the definition is by homomorphic extension:

$$\begin{array}{rll} \widetilde{\mathrm{S}} <: \widetilde{\mathrm{T}} & \text{if } \forall i{:}\ \mathrm{S}_i <: \mathrm{T}_i \\ \widetilde{\mathrm{A}}@\mathrm{K} <: \widetilde{\mathrm{B}}@\mathrm{L} & \text{if } \mathrm{K} <: \mathrm{L} \text{ and } \widetilde{\mathrm{A}} <: \widetilde{\mathrm{B}} \end{array}$$

An important property of the subtyping preorder is that it has a partial meet operator $\sqcap$.

Definition 1. *A partial binary operator $\sqcap$ on a preorder $(S, \preceq)$ is a partial meet operator if it satisfies the following for every $r,\ s,\ t \in S$:*

(a) $r \preceq t$ *and* $r \preceq s$ *imply* $t \sqcap s$ *defined and* $r \preceq t \sqcap s$
(b) $t \sqcap s$ *defined implies* $t \sqcap s \preceq t$
(c) $(t \sqcap s) \sqcap r$ *defined implies* $t \sqcap (s \sqcap r)$ *defined and* $(t \sqcap s) \sqcap r = t \sqcap (s \sqcap r)$
(d) $t \sqcap s$ *defined implies* $s \sqcap t$ *defined and* $t \sqcap s = s \sqcap t$ □

In general, because they are defined for preorders rather partial orders, partial meets are not necessarily unique. But if $\sqcap$ and $\sqcap'$ are two such operators we will always have

$$r \sqcap s <: r \sqcap' s \quad \text{and } r \sqcap' s <: r \sqcap s$$

In this paper we only define partial meets on partial orders, but we wish to remain more general so that our results can be applicable in other situations; for example when we add recursion to the language of types.

Proposition 1. *The set of types, under the subtyping preorder, has a partial meet operator.*

Proof. The operator is induced by the following equation on location types:

$$\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{A}}\} \sqcap \mathsf{loc}\{\widetilde{v}{:}\widetilde{\mathrm{B}}\} = \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{A}} \cup \widetilde{v}{:}\widetilde{\mathrm{B}}\} \quad \text{if } \forall i,j : u_i = v_j \text{ implies } \mathrm{A}_i = \mathrm{B}_j$$

For example, $\mathsf{loc}\{a{:}\mathrm{A},\ b{:}\mathrm{B}\} \sqcap \mathsf{loc}\{b{:}\mathrm{B},\ c{:}\mathrm{C}\} = \mathsf{loc}\{a{:}\mathrm{A},\ b{:}\mathrm{B},\ c{:}\mathrm{C}\}$. This is extended at other types by:

$$\begin{aligned} \mathrm{A} \sqcap \mathrm{A} &= \mathrm{A} \\ (\mathrm{S}_1, .., \mathrm{S}_n) \sqcap (\mathrm{T}_1, .., \mathrm{T}_n) &= (\mathrm{S}_1 \sqcap \mathrm{T}_1, .., \mathrm{S}_n \sqcap \mathrm{T}_n) \\ (\mathrm{A}_1, .., \mathrm{A}_n)@\mathrm{K} \sqcap (\mathrm{B}_1, .., \mathrm{B}_n)@\mathrm{L} &= [\mathrm{A}_1 \sqcap \mathrm{B}_1, .., \mathrm{A}_n \sqcap \mathrm{B}_n]@(\mathrm{K} \sqcap \mathrm{L}) \end{aligned}$$

By induction on the structure of types, one can show that this operator satisfies the claims of Definition 1. □

2.4. Standard Typing

Judgements in the typing system take three forms:

$\Gamma \vdash N$	Network N is well-formed
$\Gamma \vdash_w P$	Thread P is well-formed at location w
$\Gamma \vdash_w v{:}\mathrm{T}$	Value v is well-formed at location w with type T

Here Γ, Δ range over *type environments*, which map location names to location types and variables to base types or location types.[3] Thus environments have the form $\{\widetilde{k}{:}\widetilde{\mathrm{K}},\ \widetilde{x}{:}\widetilde{\mathrm{L}},\ \widetilde{y}{:}\widetilde{\mathsf{BT}}\}$, up to reordering. For example, the following is a type environment:

$$\Gamma = \{\ell{:}\mathsf{loc}\{a{:}\mathrm{A}, x{:}\mathrm{B}\},\ y{:}\mathsf{int},\ z{:}\mathsf{loc}\{a{:}\mathrm{A}'\}\}$$

[3] For simplicity, the typing system defined here requires that every tuple be fully decomposed upon reception; that is, terms of the form $a?(x{:}(int, int))\,P$ are not typable. The more general case is straightforward, but requires a more complex treatment of location types, as in [13].

Table III. Standard Typing

Values (rules for base values not shown):

$$\frac{\Gamma(u) <: \mathrm{T}}{\Gamma \vdash_w u{:}\mathrm{T}} \qquad \frac{\Gamma(w) <: \mathsf{loc}\{u{:}\mathrm{A}\}}{\Gamma \vdash_w u{:}\mathrm{A}} \qquad \frac{\Gamma \vdash_w u_i{:}\mathrm{T}_i \ \ (\forall i)}{\Gamma \vdash_w \widetilde{u}{:}\widetilde{\mathrm{T}}} \qquad \frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathrm{K}\\ \Gamma \vdash_u \widetilde{v}{:}\widetilde{\mathrm{B}}\end{array}}{\Gamma \vdash_w \widetilde{v}@u : (\widetilde{\mathrm{B}})@\mathrm{K}}$$

Threads:

$$\frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathsf{res}\langle \mathrm{T}\rangle\\ \Gamma \vdash_w v{:}\mathrm{T}\\ \Gamma \vdash_w P\end{array}}{\Gamma \vdash_w u!\langle v\rangle\, P} \qquad \frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathsf{res}\langle \mathrm{T}\rangle\\ \mathrm{fv}\,(X) \text{ disjoint } \mathrm{fv}\,(\Gamma)\\ \Gamma \sqcap \{_w X{:}\mathrm{T}\} \vdash_w Q\end{array}}{\Gamma \vdash_w u?(X{:}\mathrm{T})\, Q} \qquad \frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathrm{S}\\ \Gamma \vdash_w v{:}\mathrm{T}\\ \Gamma \sqcap \{_w u{:}\mathrm{T}\} \sqcap \{_w v{:}\mathrm{S}\} \vdash_w P\\ \Gamma \vdash_w Q\end{array}}{\Gamma \vdash_w \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

$$\frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathsf{loc}\\ \Gamma \vdash_u P\end{array}}{\Gamma \vdash_w \mathsf{goto}\, u.\, P} \qquad \frac{\begin{array}{l}e \notin \mathrm{fn}\,(\Gamma, w)\\ \Gamma \sqcap \{_w e{:}\mathrm{T}\} \vdash_w P\end{array}}{\Gamma \vdash_w (\nu e{:}\mathrm{T})\, P} \qquad \frac{\begin{array}{l}\Gamma \vdash_w P\\ \Gamma \vdash_w Q\end{array}}{\Gamma \vdash_w \mathsf{stop},\ P \mid Q,\ *P}$$

Networks:

$$\frac{\Gamma \vdash_k P}{\Gamma \vdash k[\![P]\!]} \qquad \frac{\begin{array}{l}e \notin \mathrm{fn}\,(\Gamma, k)\\ \Gamma \sqcap \{_k e{:}\mathrm{T}\} \vdash N\end{array}}{\Gamma \vdash (\nu_k e{:}\mathrm{T})\, N} \qquad \frac{\begin{array}{l}\Gamma \vdash M\\ \Gamma \vdash N\end{array}}{\Gamma \vdash \mathbf{0},\ M \mid N}$$

We write $\Gamma(u)$ to refer to the type of identifier u in Γ. So for Γ as defined above, $\Gamma(z) = \mathsf{loc}\{a{:}\mathrm{A}'\}$ whereas $\Gamma(u)$ is undefined.

The standard typing system is defined in Table III. This is the type system from [13, §5], with a few notational changes and the addition of base types. We implicitly assume in all rules that the environment Γ is well-formed and that each type on the right-hand-side of the turnstile is *closed*; that is, we do not allow variables to appear in location types in terms.

We presuppose a set of rules for base values, which, for example, say that integer constants have type int and the boolean constants t and f have type bool. In Table III, there are two rules for identifiers. The first applies to "universal" identifiers in the domain of the type environment: location names and variables of location or base types. The second applies to "local" identifiers in location types: resource names and variables of resource type. Universal identifiers have a consistent meaning across all sites, whereas local identifiers do not; for example, the location name ℓ refers to the same thing no matter where it occurs, whereas the resource name a does not. Note that when typing a dependent tuple $\widetilde{v}@u$, the typing of $\widetilde{v}$ is deduced with respect to the location identifier u.

For networks and threads, the main rules of interest are for agents and movement. For the agent $\ell[\![P]\!]$ to be well-typed, P must be well-typed at location ℓ; whereas for the thread $\mathsf{goto}\, u.\, P$ to be well-typed at some location w, P must be well-typed at location u.

The rules for restriction and input are intuitive, although they require some notation for environment extensions. Both subtyping and the partial meet operator extend point-wise to environments in the obvious manner: For subtyping we have:

$$\Delta <: \Gamma \text{ iff } \forall w \in \mathrm{dom}(\Gamma) : \Delta(w) <: \Gamma(w)$$

The partial meet operator $\Delta \sqcap \Gamma$ is undefined if $\Delta(w) \sqcap \Gamma(w)$ is undefined for some $w \in \mathrm{dom}(\Delta) \cap \mathrm{dom}(\Gamma)$, otherwise:

$$\begin{aligned}\Delta \sqcap \Gamma = {} & \{w{:}\mathrm{K} \mid \Delta(w) \sqcap \Gamma(w) = \mathrm{K}\} \\ & \cup \{w{:}\mathrm{K} \mid \Delta(w) = \mathrm{K} \text{ and } w \notin \mathrm{dom}(\Gamma)\} \\ & \cup \{w{:}\mathrm{K} \mid \Gamma(w) = \mathrm{K} \text{ and } w \notin \mathrm{dom}(\Delta)\}\end{aligned}$$

The following lemma ensures that this is indeed a partial meet operator:

Lemma 1.

(a) *If* $\Gamma <: \Delta_1$ *and* $\Gamma <: \Delta_2$ *then* $\Delta_1 \sqcap \Delta_2$ *defined and* $\Gamma <: \Delta_1 \sqcap \Delta_2$.
(b) *If* $\Gamma \sqcap \Delta$ *defined and* $\Delta <: \Delta'$ *then* $\Gamma \sqcap \Delta'$ *defined.* □

As typechecking with respect to an environment Γ proceeds this environment may be extended by extra type assumptions about bound variables. For example typechecking the network $(\nu_k\, e{:}\mathrm{T})\, N$ with respect to Γ requires that N is well-typed with respect to Γ augmented by the type information in the declaration $e{:}\mathrm{T}$. Because of the existence of local resouces the precise form this information takes may depend on k, where the declaration is made. We introduce the general notation, $\{_w\, u{:}\mathrm{T}\}$, where $w \in \mathit{Loc} \cup \mathit{Var}$, for this purpose. It's definition is given by induction on u and T:

$$\begin{aligned}\{_w\, \mathsf{bv}{:}\mathsf{BT}\} &= \varnothing, \quad \text{if } \mathsf{bv} \in \mathrm{valset}(\mathsf{BT}) \\ \{_w\, x{:}\mathsf{BT}\} &= \{\, x{:}\mathsf{BT}\} \\ \{_w\, k{:}\mathrm{K}\} &= \{\, k{:}\mathrm{K}\} \\ \{_w\, x{:}\mathrm{K}\} &= \{\, x{:}\mathrm{K}\} \\ \{_w\, a{:}\mathrm{A}\} &= \{w{:}\mathsf{loc}\{a{:}\mathrm{A}\}\} \\ \{_w\, x{:}\mathrm{A}\} &= \{w{:}\mathsf{loc}\{x{:}\mathrm{A}\}\} \\ \{_w\, \widetilde{v}@u{:}(\widetilde{\mathrm{B}})@\mathrm{K}\} &= \{_w\, u{:}\mathrm{K}\} \sqcap \{_u\, \widetilde{v}{:}\widetilde{\mathrm{B}}\} \\ \{_w\, \widetilde{u}{:}\widetilde{\mathrm{T}}\} &= \{_w\, u_1{:}\mathrm{T}_1\} \sqcap \cdots \sqcap \{_w\, u_n{:}\mathrm{T}_n\}\end{aligned}$$

For example:

$$\begin{aligned}\{_w\, (0, a){:}(\mathsf{int}, \mathrm{A})\} &= \{w{:}\mathsf{loc}\{a{:}\mathrm{A}\}\} \\ \{_w\, (k, k@c{:}(\mathsf{loc}\{a{:}\mathrm{A}\}, \mathsf{loc}\{b{:}\mathrm{B}\}[\mathrm{C}])\} &= \{k{:}\mathsf{loc}\{a{:}\mathrm{A}, b{:}\mathrm{B}, c{:}\mathrm{C}\}\}\end{aligned}$$

To understand the notation, the reader may wish to consider the following results, which are straightforward to establish.

Lemma 2.

(a) *If* $\Gamma \vdash_w u{:}\mathrm{T}$ *then* $\{_w u{:}\mathrm{T}\}$ *defined and* $\Gamma <: \{_w u{:}\mathrm{T}\}$.
(b) *If* $\{_w u{:}\mathrm{S}\} \sqcap \{_w u{:}\mathrm{T}\}$ *defined then* $\{_w u{:}(\mathrm{S} \sqcap \mathrm{T})\}$ *defined.* □

With this notation the rule for restriction in networks, for example, should be easily understandable. The network $(\nu_k\, e{:}\mathrm{T})\, N$ is well-typed with respect to Γ, $\Gamma \vdash (\nu_k\, e{:}\mathrm{T})\, N$, if e is new to Γ and k, and N is well-typed with respect to Γ extended at k by the type information in declaration $e{:}\mathrm{T}$, that is, $\Gamma \sqcap \{_k\, e{:}\mathrm{T}\} \vdash N$.

The rule for matching allows the combination of capabilities available on different instances of a location name. Note that the rule may only be applied when $\mathrm{S} \sqcap \mathrm{T}$ is defined. In the case that $\mathrm{S} = \mathrm{T}$, the rule degenerates to the standard rule for conditionals:

$$\frac{\Gamma \vdash_w u{:}\mathrm{T},\ v{:}\mathrm{T},\ P,\ Q}{\Gamma \vdash_w \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

The extra generality of the rule is necessary to type threads such as the following:

$$a?(x@z)\ \ b?(y@w)\ \ \mathsf{if}\ z = w\ \mathsf{then}\ \mathsf{goto}\, z.\, (x?(u)\ \ y!\langle u\rangle\,)$$

This thread receives two remote channels from different sources, then forwards messages from one channel to the other. Further examples are given in [13] where we argue that the more general rule is crucial for typing many practical applications.

The typing system satisfies several standard properties such as type specialisation, weakening and a substitution lemma, as described in [13]. The following result establishes that well-typed terms are free of runtime errors throughout their execution.

Theorem 1 (Subject Reduction for the Standard Semantics).

If $\Gamma \vdash N$ *and* $N \longrightarrow N'$ *then* $\Gamma \vdash N'$.

Proof. See [13, Theorem 5.1]. □

3. Partial Typing

The purpose of this paper is to study systems in which only a subset of agents are known to be well typed. Since agents themselves are unnamed and can move about the network, we draw the distinction between the typed and the untyped worlds using *locations*, or *sites*.

In this section we first define a *partial typing system* which allows agents at certain *untyped*, or *bad*, locations to have arbitrary, potentially malicious behaviour. We then present an example which shows that the standard semantics is inadequate for partially typed systems and finally point to the solution proposed in later sections.

3.1. The Partial Typing Relation

To capture the notion of a *untyped* locations formally, we introduce a new location type, lbad, into the type language. We use the terms *untyped* and *bad* interchangeably, similarly *typed* and *good*. Location types are now defined:

$$\mathrm{K}, \mathrm{L} ::= \mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}, \widetilde{x}{:}\widetilde{\mathrm{B}}\} \;\mid\; \mathsf{lbad}$$

We sometimes refer to types in the augmented language as *partial types*. The subtype relation is extended to partial types by adding the following subtyping rule:

$$\mathsf{lbad} \mathrel{<:} \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{A}}\}$$

This reflects the fact that channels at an untyped location may have any type and consequently behaviour at bad locations is unconstrained. With the addition of lbad, the partial meet operator becomes total on location types.

Proposition 2. *The set of types, extended with* lbad*, under the subtyping preorder, has a partial meet operator.*

Proof. It is straightforward to show that the following definition provides an extension of the partial meet operator defined in 1:

$$\begin{aligned}
\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \sqcap \mathsf{loc}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} &= \begin{cases} \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}} \cup \widetilde{v}{:}\widetilde{\mathrm{T}}\} & \text{if } \forall i, j : u_i = v_j \text{ implies } \mathrm{S}_i = \mathrm{T}_j \\ \mathsf{lbad} & \text{otherwise} \end{cases} \\
\mathsf{lbad} \sqcap \mathsf{loc}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} &= \mathsf{lbad} \\
\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{T}}\} \sqcap \mathsf{lbad} &= \mathsf{lbad}
\end{aligned}$$

□

The typing relation given in Table III, $\Gamma \vdash N$, may now be applied to this extended language of types with the result that untyped locations enjoy many expected properties. For example, since $\mathsf{lbad} \mathrel{<:} \mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}$ and $\mathsf{lbad} \mathrel{<:} \mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{bool}\rangle\}$, we can infer

$$\{m{:}\mathsf{lbad}\} \vdash_m (a, a){:}(\mathsf{res}\langle\mathsf{int}\rangle, \mathsf{res}\langle\mathsf{bool}\rangle)$$

In general we can infer that a resource at an untyped location has any resource type, meaning that local computations at these locations are

Table IV. Partial Typing Relation

All rules from III but those for restriction (ν)

$$\textsf{(thread-bad)}\frac{\Gamma(w) = \mathsf{lbad}}{\Gamma \vdash_w P} \qquad \textsf{(thread-new}_\mathsf{g}\textsf{)}\frac{\begin{array}{l}\mathrm{T} \neq \mathsf{lbad}\\ e \notin \mathrm{fn}(\Gamma, w)\\ \Gamma \sqcap \{_w\, e{:}\mathrm{T}\} \vdash_w P\end{array}}{\Gamma \vdash_w (\nu\, e{:}\mathrm{T})\, P}$$

$$\textsf{(net-new}_\mathsf{b}\textsf{)}\frac{\begin{array}{l}\Gamma(k) = \mathsf{lbad}\\ \ell \notin \mathrm{fn}(\Gamma, k)\\ \Gamma \sqcap \{\ell{:}\mathsf{lbad}\} \vdash N\end{array}}{\Gamma \vdash (\nu_k \ell{:}\mathrm{L})\, N} \qquad \textsf{(net-new}_\mathsf{g}\textsf{)}\frac{\begin{array}{l}\mathrm{T} \neq \mathsf{lbad}\\ e \notin \mathrm{fn}(\Gamma, k)\\ \Gamma \sqcap \{_k\, e{:}\mathrm{T}\} \vdash N\end{array}}{\Gamma \vdash (\nu_k\, e{:}\mathrm{T})\, N}$$

unconstrained by typing considerations. This is the case even if the resource is restricted. For example if $\Gamma(k) = \mathsf{lbad}$ then, since $\mathsf{lbad} \sqcap \mathsf{loc}\{a{:}\mathrm{A}\} = \mathsf{lbad}$, the judgement $\Gamma \vdash (\nu_k\, a{:}\mathrm{A})\, N$ follows from $\Gamma \vdash N$. Moreover, since $\mathsf{lbad} <: \mathsf{loc}\{a{:}\mathrm{B}\}$ for any resource type B, in this latter type judgement the type of occurrences of a in N may be arbitrary. Note however that, because of our separate categories for base, resource and location identifiers, no matter what the environment we cannot infer $a{:}\mathsf{loc}$, $k{:}\mathsf{int}$ or $2{:}\mathsf{res}\langle\rangle$.

Agents can also use the type information to infer that a remote location is untyped. For example consider an environment Γ such that:

$$\Gamma(\ell) = \mathsf{loc}\left\{\begin{array}{l} b{:}\mathsf{res}\langle\mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{bool}\rangle\}\rangle\\ c{:}\mathsf{res}\langle\mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}\rangle\\ d{:}\mathsf{res}\langle\mathsf{lbad}\rangle\end{array}\right\}$$

Then the network

$$\ell[\![b?(z)\ \ c?(w)\ \ \mathsf{if}\ z = w\ \mathsf{then}\ d!\langle z\rangle]\!]$$

is well-typed with respect to Γ. If the same location m is received on both the channels b and c, then the agent knows that m is untyped. Thus m can be output on d, a channel that transmits locations of the type lbad.

Despite these examples, the standard typing system does not quite capture the notion of "untyped location", even with the addition of lbad. Most important, the standard typing rule for movement does not allow untyped locations to send malicious agents to typed locations. Consider a type environment Γ, defined as:

$$\Gamma = \left\{\begin{array}{l} k : \mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}\\ m : \mathsf{lbad}\end{array}\right\}$$

We would like to have that $\Gamma \vdash m[\![\mathsf{goto}\, k.\, a!\langle \mathsf{t}\rangle]\!]$. Here an untyped agent at m attempts to move to k and misuse the channel a. The standard typing rule for movement, however, does not allow this judgement, since the standard rule for movement requires that $a!\langle \mathsf{t}\rangle$ be well-typed at k, which definitely is not the case.

The *partial typing relation* is defined in Table IV. All of the rules of the standard type system carry over to the partial typing system but for those concerning restriction, which require an additional side condition. Most important, the introduction of the rule (thread-bad) allows untyped locations to have truly arbitrary behaviour, including the ability to (attempt to) send malicious agents to good locations. Thus the partial typing relation validates the judgement $\Gamma \vdash m[\![\mathsf{goto}\, k.\, a!\langle \mathsf{t}\rangle]\!]$, with Γ as given in the previous paragraph.

The rule ($\mathsf{net\text{-}new_b}$) says that locations created at untyped locations should themselves be untyped. This rule is required to maintain well-typing under reductions such as:

$$k[\![(\nu\ell{:}\mathrm{L})\,\mathsf{goto}\,\ell.\,P]\!] \longmapsto (\nu_k\ell{:}\mathrm{L})\,k[\![\mathsf{goto}\,\ell.\,P]\!] \longmapsto (\nu_k\ell{:}\mathrm{L})\,\ell[\![P]\!]$$

The rules ($\mathsf{thread\text{-}new_g}$) and ($\mathsf{net\text{-}new_g}$) are as in the standard type system, but require that typed locations not create untyped ones. This "reasonableness requirement" is necessary to establish Type Safety, as formulated in Theorem 3.

3.2. An Example

Consider the following (partial) type environment:

$$\Gamma = \left\{ \begin{array}{l} k : \mathsf{loc}\,\{a : \mathsf{res}\langle\mathsf{int}\rangle\} \\ \ell : \mathsf{loc}\left\{ \begin{array}{l} b : \mathsf{res}\langle\mathsf{res}\langle\mathsf{bool}\rangle@\mathsf{loc}\rangle \\ c : \mathsf{res}\langle\mathsf{res}\langle\mathsf{int}\rangle@\mathsf{loc}\rangle \end{array}\right\} \\ m : \mathsf{lbad} \end{array} \right\}$$

Here we have three locations, k, ℓ and m, the first two of which are typed, and the last untyped. Of the good (typed) sites, we know that k has an integer channel a, and ℓ has two channels: c, which communicates located integer channels; and b, which communicates located boolean channels.

Consider a system with two agents at ℓ, waiting to receive data on channels c and b, respectively. The first agent will expect, as the second element of the tuple it receives, the name of an integer channel, whereas the second will expect the name of a boolean channel. In addition suppose that there are agents at k and m poised to send data to ℓ on channels c and b, respectively. Such a system is the following:

$$\begin{aligned} N = \quad & \ell[\![c?\langle y@w\rangle \;\; \mathsf{goto}\, w.\, y!\langle 0\rangle]\!] \\ & \mid \ell[\![b?\langle x@z\rangle \quad \mathsf{goto}\, z.\, x!\langle \mathsf{t}\rangle]\!] \\ & \mid k[\![\mathsf{goto}\, \ell.\, c!\langle a@k\rangle]\!] \\ & \mid m[\![\mathsf{goto}\, \ell.\, b!\langle a@k\rangle]\!] \end{aligned}$$

Here the agents at ℓ and k are all quite reasonable; they could be typed using the standard type system of Table III. The final agent, at m, however, flagrantly violates the types of channels a and b; this agent intends to send an integer channel (a) where a boolean channel is expected (on b).

One can easily see that, using the standard typing system (without lbad), for no Δ do we have $\Delta \vdash N$. This is because channel a at k may be bound to either y or x, and these identifiers are subject to conflicting uses. There is no assignment of standard types to a, b and c that satisfies all of the constraints given in N. On the other hand, using the partial typing system, we have $\Gamma \vdash N$. This well typing, however, is not preserved by reduction.

First consider the agents communicating on c. Using standard reduction, as defined in Table II, these agents reduce as follows.

$$\begin{aligned} & \ell[\![c?\langle y@w\rangle \;\; \mathsf{goto}\, w.\, y!\langle 0\rangle]\!] \mid k[\![\mathsf{goto}\, \ell.\, c!\langle a@k\rangle]\!] && (1) \\ \longrightarrow\; & \ell[\![c?\langle y@w\rangle \;\; \mathsf{goto}\, w.\, y!\langle 0\rangle]\!] \mid \ell[\![c!\langle a@k\rangle]\!] && (2) \\ \longrightarrow\; & \ell[\![\mathsf{goto}\, k.\, a!\langle 0\rangle]\!] && (3) \\ \longrightarrow\; & k[\![a!\langle 0\rangle]\!] && (4) \end{aligned}$$

The first reduction, (1) to (2) follows from (r-move), (2) to (3) from (r-comm) and (s-garbage), and (3) to (4) from (r-move) again. All of these reductions preserve well-typing under Γ.

Now consider the agents communicating on b.

$$\begin{aligned} & \ell[\![b?\langle x@z\rangle \;\; \mathsf{goto}\, z.\, x!\langle \mathsf{t}\rangle]\!] \mid m[\![\mathsf{goto}\, \ell.\, b!\langle a@k\rangle]\!] && (5) \\ \longrightarrow\; & \ell[\![b?\langle x@z\rangle \;\; \mathsf{goto}\, z.\, x!\langle \mathsf{t}\rangle]\!] \mid \ell[\![b!\langle a@k\rangle]\!] && (6) \\ \longrightarrow\; & \ell[\![\mathsf{goto}\, k.\, a!\langle \mathsf{t}\rangle]\!] && (7) \\ \longrightarrow\; & k[\![a!\langle \mathsf{t}\rangle]\!] && (8) \end{aligned}$$

The reductions are derived just as before, but (6), (7) and (8) are not well-typed under Γ. This fact is obvious when considering (8) where an agent at k attempts to send a boolean on an integer channel. Already in (6), however, typing under Γ fails. In order to infer $\Gamma \vdash \ell[\![b!\langle a@k\rangle]\!]$ we must establish that for some T, $\Gamma \vdash_{\ell} b{:}\mathsf{res}\langle \mathrm{T}\rangle$ and $\Gamma \vdash_{\ell} a@k{:}\mathrm{T}$.

Given the type of b at ℓ, we would have to take $\mathrm{T} = \mathsf{res}\langle\mathsf{bool}\rangle@\mathsf{loc}$, but $\Gamma \nvdash_{\ell} a@k{:}\mathsf{res}\langle\mathsf{bool}\rangle@\mathsf{loc}$, since a is an integer channel at k.

The semantics presented in the following section will prevent the reduction of (5) to (6) by *dynamically* typing certain agents when they move from one location to another. To accomplish this, we augment the standard reduction semantics with type information detailing the resources available at each site. Significantly, this type information is held *locally* at each site, and thus sites will have different *views* of the network. Crucial to this semantics is the ability of a location to determine the *authority* of an incoming thread, that is, the location from which the thread was sent. This semantics is improved in Section 5 by adding *trusted locations* to the type system. In each of these sections, the main results are Subject Reduction (for the partial typing relation) and Type Safety.

It is worth contrasting this approach with the "purely local" approach adopted for "anonymous networks" in [12] (and outlined in Section B). In anonymous networks, the authority of incoming threads is not known. The semantics of [12] uses a weaker typing system requiring consistency only of *local* resource types. Thus, in that work, (6) is taken to be well-typed, with subject reduction failing only in the move from (7) to (8). The chief advantage of the current work is that it permits the use of *trust*, which appears to be incompatible with terms such as (6).

4. Filters and Authorities

In this section we propose a semantics which recovers subject reduction for partially-typed networks. The solution assumes that the origin, or *authority*, of incoming agents can be reliably determined.

4.1. Syntax and Semantics

To accomplish dynamic typechecking, it is necessary to add type information to running networks. We do this by adding a *filter* $k\langle\!\langle\Delta\rangle\!\rangle$ for each location k in a network. The filter includes a type environment Δ which gives k's view of the resources in the network. Suppose that in a network N, location k knows that there is resource named a of type A at location ℓ. This intuition is captured by requiring that N have a subterm $k\langle\!\langle\Delta\rangle\!\rangle$ such that $\Delta(\ell) <: \mathsf{loc}\{a{:}\mathrm{A}\}$.

Formally, we extend the syntax of networks in Table I to include filters, as follows:

$$N \ ::= \ \ldots \ \mid \ k\langle\!\langle\Delta\rangle\!\rangle$$

We say that a term $k\langle\!\langle\Delta\rangle\!\rangle$ is a *filter for* k. The typing and reduction relations for networks with filters are given in Table V.

Static Typing: The static typing relation extends that of Table III and Table IV with the two rules, given in Table V. The rule ($\mathsf{net\text{-}filter_g}$) requires that a filter for a good location k must have full knowledge of the resources at k ($\Gamma(k) = \Delta(k)$) and a view of the rest of the world that is consistent with reality ($\Gamma <: \Delta$). The rule ($\mathsf{net\text{-}filter_b}$) indicates that filters for bad locations may be arbitrary.

These typing rules guarantee that whenever a filter exists, it must have a reasonable view of the world, but the rules do not constrain the number of filters for a given location. We could extend the type system to guarantee that each location have a unique filter, but we prefer to impose this constraint outside the typing relation.

Definition 2. *We say that a network N is* well formed *if for every $k \in \mathrm{fn}(N)$ there is exactly one subterm of N which is a filter for k, and for every subterm $(\nu_m \ell{:}\mathrm{L})\, M$ of N there is exactly one subterm of M which is a filter for ℓ.* □

For the rest of the paper, we consider only well-formed networks.

Reduction: As networks evolve, a site's filter should be augmented to reflect its increasing knowledge of the network. At the very least this should include updates with information about new local resources. The rule ($\mathsf{r_f\text{-}newr}$) says that when a new resource a is created at k, the type of that resource is recorded in the filter for k. This ensures that k continues to have full knowledge of local resources. Similarly when a new location ℓ is created by k, a new filter should be created for ℓ and the filter for k updated to establish a view of ℓ. This is achieved by the rule ($\mathsf{r_f\text{-}newl}$).

In addition, filters may take other measures to increase their knowledge of the network. Here we modify the communication rule as follows: when a value is received at a site, the site's filter is augmented to include any new information that can be gleaned from the communicated value. This is reflected in the rule ($\mathsf{r_f\text{-}comm}$).

The purpose of filters is to check that incoming agents are well-typed. Thus, the main change to the semantics is replace the reduction rule ($\mathsf{r\text{-}move}$) with:

$$\ell\langle\!\langle\Delta\rangle\!\rangle \;|\; k[\![\mathsf{goto}\,\ell.\,P]\!] \longmapsto \ell\langle\!\langle\Delta\rangle\!\rangle \;|\; \ell[\![P]\!] \quad \text{if } k=\ell \text{ or } \Delta \Vdash^{k}_{\ell} P$$

Here $\Delta \Vdash^{k}_{\ell} P$ is a *dynamic typing relation*, which intuitively says that P is well-formed to move to location ℓ, if acting under *authority of* k.

Table V. Typing and reduction using filters

Static typing: all rules from IV

$$(\textsf{net-filter}_\textsf{g})\,\frac{\Gamma <: \Delta \qquad \Gamma(k) = \Delta(k)}{\Gamma \vdash k\langle\!\langle \Delta \rangle\!\rangle} \qquad\qquad (\textsf{net-filter}_\textsf{b})\,\frac{\Gamma(k) = \textsf{lbad}}{\Gamma \vdash k\langle\!\langle \Delta \rangle\!\rangle}$$

Reduction precongruence: (r-split), ($\textsf{r-eq}_1$) and ($\textsf{r-eq}_2$) rules for $\equiv$ from II

$$
\begin{array}{lll}
(\textsf{r}_\textsf{f}\textsf{-move}) & k[\![\textsf{goto}\,\ell.\,P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle & \text{if } k = \ell \text{ or } \Delta \Vdash^{k}_{\ell} P \\
& \longmapsto \quad \ell[\![P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle & \\[1ex]
(\textsf{r}_\textsf{f}\textsf{-newr}) & k[\![(\nu\, a{:}\mathrm{A})\, P]\!] \mid k\langle\!\langle \Delta \rangle\!\rangle & \text{if } a \notin \mathrm{fn}(\Delta) \\
& \longmapsto (\nu_k\, a{:}\mathrm{A})\,\big(k[\![P]\!] \mid k\langle\!\langle \Delta \sqcap \{{}_k\, a{:}\mathrm{A}\} \rangle\!\rangle\big) & \\[1ex]
(\textsf{r}_\textsf{f}\textsf{-newl}) & k[\![(\nu\ell{:}\mathrm{L})\, P]\!] \mid k\langle\!\langle \Delta \rangle\!\rangle & \text{if } \ell \notin \mathrm{fn}(\Delta) \cup \{k\} \\
& \longmapsto (\nu_k \ell{:}\mathrm{L})\,\big(k[\![P]\!] \mid k\langle\!\langle \Delta \sqcap \{\ell{:}\mathrm{L}\} \rangle\!\rangle \mid \ell\langle\!\langle \{\ell{:}\mathrm{L}\} \rangle\!\rangle\big) & \\[1ex]
(\textsf{r}_\textsf{f}\textsf{-comm}) & k[\![a!\langle v\rangle\, P]\!] \mid k[\![a?(X{:}\mathrm{T})\, Q]\!] \mid k\langle\!\langle \Delta \rangle\!\rangle & \\
& \longmapsto \quad k[\![P]\!] \mid k[\![Q\{\!\!\{ v/X \}\!\!\}]\!] \mid k\langle\!\langle \Delta \sqcap \{{}_k\, v{:}\mathrm{T}\} \rangle\!\rangle &
\end{array}
$$

Dynamic typing: all rules from IV, with '$\Vdash^{k}_{w}$' replacing '$\vdash_w$'

$$(\textsf{val}_\textsf{f}\textsf{-self}_1)\,\frac{\textsf{lbad} <: \mathrm{K}}{\Delta \Vdash^{k}_{w} k{:}\mathrm{K}} \qquad (\textsf{val}_\textsf{f}\textsf{-self}_2)\,\frac{}{\Delta \Vdash^{k}_{k} a{:}\mathrm{A}} \qquad (\textsf{thread}_\textsf{f}\textsf{-return})\,\frac{}{\Delta \Vdash^{k}_{w} \textsf{goto}\, k.\, P}$$

Agents originating locally are assumed to be well-typed and therefore need not be checked dynamically.

Dynamic Typing: One approach to dynamic typing would be to take the dynamic typing relation to be the same as the static typing relation: $(\Vdash^{k}_{w}) = (\vdash_w)$. In effect, this would limit incoming agents to include only names of resources that are known in advance. While this is certainly sound, it is much too restrictive; for example, new resources could only be used by agents that originated locally. Consider the system:

$$k[\![(\nu a)\, \textsf{goto}\, \ell.\, b!\langle a@k\rangle]\!] \;\mid\; \ell[\![b?(x@z)\, P]\!] \;\mid\; \ell\langle\!\langle \Delta \rangle\!\rangle \qquad (*)$$

Here k creates a new resource and wishes to communicate it to ℓ. However with $(\Vdash^{k}_{w}) = (\vdash_w)$ the move from k to ℓ is refused — ($\textsf{r}_\textsf{f}\textsf{-move}$) cannot be applied — since the filter Δ at ℓ can have no knowledge of the new resource a.

At the opposite extreme, we might allow threads to include any reference to non-local resources. However, this approach is clearly unsound from the counter-example given in the last section. The difficulty is that threads from bad locations may provide incorrect information about good locations, breaking subject reduction.

To straddle the gap between sound-but-useless and unsound-but-expressive, we introduce the notion of *authority*. We say that an agent leaving a location k acts *under the authority of* k. When an agent with authority k enters another location, we say that k is the *authority* of the agent.

While it is not safe to allow incoming agents to refer to *any* non-local resources, it is safe to allow them to refer to resources located at their authority, that is, at their "home" location. Intuitively this is true because, under this discipline, "bad" agents can only to "lie" about resources located at their authority, which must have been a bad location to begin with. Lies about bad locations do not hurt well-typing, since bad locations are untyped. This discipline is quite general and should be applicable in many frameworks. Here we show how it can be formalised using a set of runtime typing rules.

Formally, these rules extend those of the static type system given in Table III and Table IV with two additional rules for values and one for threads. These rules allow references to an incoming agent's authority to go unchecked. The rule ($\mathsf{val_f}$-$\mathsf{self_1}$) allows an incoming agent to refer to its authority k, regardless of whether the filter environment Δ contains any information about k. (Note that the condition $\mathsf{lbad} <: \mathrm{K}$ is vacuously satisfied; we include it here only for reference in the next section.) The rule ($\mathsf{val_f}$-$\mathsf{self_2}$) allows an incoming agent to refer to resources at its authority. To see the effect of these rules let us consider a simple example.

Example 2. *Let* $\Delta_\ell = \{\ell{:}\mathsf{loc}\{a{:}\mathsf{res}\langle \mathrm{B}\,@\mathrm{K}\rangle\}\}$ *for some appropriate types* K *and* B*. First note that we can not infer that* $\Delta_\ell \vdash_\ell a!\langle b\,@k\rangle$ *using the static typing system, because* k *is not even mentioned in* Δ_l*. However we can deduce* $\Delta_\ell \Vdash_\ell^k a!\langle b\,@k\rangle$ *using the dynamic typing relation. This follows by an application of the output rule in Table III to the two judgements*

1. $\Delta_\ell \Vdash_\ell^k a{:}\mathsf{res}\langle \mathrm{B}\,@\mathrm{K}\rangle$
2. $\Delta_\ell \Vdash_\ell^k b\,@k{:}\mathrm{B}\,@\mathrm{K}$

The first follows by the standard typing rules in Table III while the second requires the new dynamic typing rules from Table V. First an application of ($\mathsf{val_f}$-$\mathsf{self_1}$) *allows the establishment of*

$$\Delta_\ell \Vdash_\ell^k k{:}\mathrm{K}$$

regardless of the actual type K*. Incoming threads are allowed to state arbitrary properties of their source location. Then an application of* ($\mathsf{val_f}$-$\mathsf{self_2}$) *gives*

$$\Delta_\ell \Vdash_k^k b{:}\mathrm{B}$$

again regardless of the form of B*. So an application of the rule for located names from Table III gives the required (2) above.*

Thus the following reduction is allowed by the semantics:

$$k[\![\mathsf{goto}\,\ell.\ a!\langle k,b\rangle]\!] \ \mid\ \ell\langle\!\langle \Delta_\ell \rangle\!\rangle \longrightarrow \ell[\![a!\langle k,b\rangle]\!] \ \mid\ \ell\langle\!\langle \Delta_\ell \rangle\!\rangle$$

□

The rule ($\mathsf{thread_f\text{-}return}$) allows a thread to return to its home location without subjecting the returning thread to further typechecking. This rule allows some additional expressiveness and reduces the burdens of typechecking somewhat.

Note that while the static typing system interprets the rules of Table III and Table IV with respect to an omniscient authority (Γ), the dynamic type system interprets these rules with respect to the knowledge contained in a filter (Δ, where $\Gamma <: \Delta$). Whereas untypability with respect to Γ indicates that a network is malformed, untypability with respect to Δ may simply indicate that Δ has insufficient information to determine whether an agent is malicious or not.

The fact that resource names are not assigned unique locations is crucial to the success of our strategy for dynamic typechecking. It would be difficult to see how to formulate our approach while maintaining the assumption that each name had a unique location (as, for example, in [2]). For example, suppose that the resource a was "uniquely located" at k. Then the agent $m[\![\mathsf{goto}\,\ell.\ b!\langle a@m\rangle]\!]$ at the bad site m could "hijack" a using ($\mathsf{thread\text{-}self_2}$), convincing ℓ that a was uniquely located at m, rather than some good location k. In particular entry to ℓ by an agent from k may subsequently be blocked because ℓ mistakenly believes that the unique location of a is at m.

4.2. Examples

Example 3. *First we show how filters are updated via communication with imported agents. Consider the network* (*) *discussed above, where the location k wishes to transmit to ℓ the name of a new resource a. If $\Delta = \{\ell{:}\mathsf{loc}\{b{:}\mathsf{res}\langle \mathrm{A}@\mathrm{K}\rangle\}\}$ then we have the following reductions:*

$$k[\![(\nu a)\,\mathsf{goto}\,\ell.\ b!\langle a@k\rangle]\!] \mid \ell[\![b?(x@z{:}\mathrm{A}@\mathrm{K})\ P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle \longrightarrow^{*} (\nu_k\, a)\ (\ell[\![P']\!] \mid \ell\langle\!\langle \Delta' \rangle\!\rangle)$$

where $P' = P\{\!\!\{^{k,a}/_{z,x}\}\!\!\}$ *and* $\Delta' = \Delta \sqcap \{k{:}\mathsf{loc}\{a{:}\mathsf{res}\langle \mathrm{A}\rangle\}\}$*. After the communication, the filter for ℓ contains information about the type of resource a at k.* □

Example 4. *Let us now revisit network* (5) *discussed in Section 3.2, which shows that partial typing is not preserved by the standard reduction relation. To use the new semantics, we must add a filter for each* *location. Here we show only the filter for* ℓ, $\ell\langle\!\langle\Delta\rangle\!\rangle$, *where* Δ *satisfies the constraints of* $(\mathsf{net\text{-}filter_g})$. *Thus, let us consider the network*

$$\Gamma \vdash m[\![\mathsf{goto}\,\ell.\, b!\langle a\,@k\rangle]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle$$

where Γ *is as given in Section 3.2. Note that the agent at* m *attempts to misinform an agent at* ℓ *about the type of the resource* a *at* k. *In the revised reduction semantics the move from* m *to* ℓ *is allowed only if* $\Delta \Vdash^m_\ell b!\langle a\,@k\rangle$, *that is if we can dynamically typecheck* $b!\langle a\,@k\rangle$ *using the filter* Δ *under the authority* m. *But this is impossible, since* $\Gamma \vdash \ell\langle\!\langle\Delta\rangle\!\rangle$. *To see this, first note that* ℓ *has full self-knowledge, that is,* $\Delta(\ell) = \Gamma(\ell)$, *and therefore* $\Delta(\ell)$ *must have the entry* $b : \mathsf{res}\langle\mathsf{res}\langle\mathsf{bool}\rangle\,@\mathsf{loc}\rangle$; *therefore to type the term we must be able to deduce* $\Delta \Vdash^m_k a{:}\mathsf{res}\langle\mathsf{bool}\rangle$. *Next note that* Δ *must be consistent with reality, namely* Γ. *This means that if* Δ *has knowledge of the resource* a *at* k *then it must be at the conflicting type* $\mathsf{res}\langle\mathsf{int}\rangle$; *therefore the rules of Table III cannot be used to infer* $\Delta \Vdash^m_k a{:}\mathsf{res}\langle\mathsf{bool}\rangle$. *Finally, since* k *is not the authority of the thread, neither can the additional rules of Table V be used to justify the claim that* $\Delta \Vdash^m_k a{:}\mathsf{res}\langle\mathsf{bool}\rangle$. *It follows that the inference* $\Delta \Vdash^m_\ell b!\langle a\,@k\rangle$ *is impossible.* □

Example 5. *Let us now modify the previous example so that* m *attempts to relate information about its* own *resources, rather than those of* k. *In such cases, movement always succeeds, whether or not the source site is bad. For example, we have the reduction:*

$$m[\![\mathsf{goto}\,\ell.\, b!\langle a\,@m\rangle]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle \;\longrightarrow\; \ell[\![b!\langle a\,@m\rangle]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle$$

This follows since $\Delta \Vdash^m_\ell a\,@m{:}\mathsf{res}\langle\mathsf{bool}\rangle\,@\mathsf{loc}$ *can be inferred using dynamic typechecking. First* $(\mathsf{val_f\text{-}self_1})$ *gives* $\Delta \Vdash^m_\ell m{:}\mathsf{loc}$ *and* $(\mathsf{val_f\text{-}self_2})$ *gives* $\Delta \Vdash^m_m a{:}\mathsf{res}\langle\mathsf{bool}\rangle$ *and so the standard rule for located values gives the required inference* $\Delta \Vdash^m_\ell a\,@m{:}\mathsf{res}\langle\mathsf{bool}\rangle\,@\mathsf{loc}$. *Note that this is regardless of the actual type assigned to* m *in* Δ. □

Example 6. *A untyped site will also succeed in sending an agent if the reception site already knows the information being received. For example suppose the view of* ℓ *is increased so that it now contains the resource* a *at* k, *that is* $\Delta(k) = \mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}$. *Then we have the reduction*

$$m[\![\mathsf{goto}\,\ell.\, c!\langle a\,@k\rangle]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle \longrightarrow \ell[\![c!\langle a\,@k\rangle]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle$$

because of the inference $\Delta \Vdash^m_\ell c!\langle a\,@k\rangle$. *Of course the authority of* m *plays no role in this judgement.* □

Example 7. *The information in filters determine which migrations are allowed, and reductions in turn may increase the information in filters. This means that certain migrations can remain blocked until the appropriate filter has been updated.*

Consider the following network, again typed using the environment Γ given in Section 3.2, where Δ is the restriction of Γ onto ℓ, that is, $\Delta = \{\ell{:}\Gamma(\ell)\}$:

$$m[\![\mathsf{goto}\,\ell.\, c!\langle a\,@k\rangle]\!] \;|\; k[\![\mathsf{goto}\,\ell.\, c!\langle a\,@k\rangle]\!] \;|\; \ell[\![*c?(x\,@z)\,P]\!] \;|\; \ell\langle\!\langle\Delta\rangle\!\rangle$$

Here the migration from m to ℓ is not immediately possible, since $\Delta \not\Vdash_{\ell}^{m} c!\langle a\,@k\rangle$; the location k is not known to Δ and the extra rules ($\mathsf{val_f}$-$\mathsf{self_1}$) and ($\mathsf{val_f}$-$\mathsf{self_2}$) can not be used because the authority (that is the source of the migration) is not m. However the migration from k is allowed since $\Delta \Vdash_{\ell}^{k} c!\langle a\,@k\rangle$ can be inferred using these rules. So the network reduces, after communication on c, to:

$$m[\![\mathsf{goto}\,\ell.\, c!\langle a\,@k\rangle]\!] \;|\; \ell[\![P']\!] \;|\; \ell[\![*c?(x\,@z)\,P]\!] \;|\; \ell\langle\!\langle\Delta'\rangle\!\rangle$$

where $P' = P\{\!|^{k,a}\!/_{z,x}|\!\}$ and $\Delta' = \Delta \sqcap \{k{:}\mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}\}$ is obtained from Δ by updating the entry for k. The migration from m to ℓ can now take place since the new guardian Δ' now knows about k. In fact $\Delta' \Vdash_{\ell}^{m} a\,@k{:}\mathsf{res}\langle\mathsf{bool}\rangle\,@\mathsf{loc}$ using the standard typing rules in Table III, from which $\Delta' \Vdash_{\ell}^{m} c!\langle a\,@k\rangle$ then follows.

This allows the network to reduce, after a further communication, to:

$$\ell[\![P']\!] \;|\; \ell[\![P']\!] \;|\; \ell[\![*c?(x\,@z)\,P]\!] \;|\; \ell\langle\!\langle\Delta'\rangle\!\rangle$$

In the absence of other agents, the migrations can only be executed in one order (k first). □

Example 8. *As a filter is updated, contradictory evidence may be obtained about a site, in which case the site must be untyped and can safely be deemed to be bad. As an example let Γ and the filter $\Delta = \{\ell{:}\Gamma(\ell)\}$ be as before, and consider the network:*

$$m[\![\mathsf{goto}\,\ell.\, b!\langle d\,@m\rangle\; c!\langle d\,@m\rangle]\!] \;|\; \ell[\![b?(x\,@z)\; c?(y\,@w)\,P]\!] \;|\; \ell\langle\!\langle\Delta\rangle\!\rangle$$

After the migration from m to ℓ and one communication this reduces to

$$\ell[\![c!\langle d\,@m\rangle]\!] \;|\; \ell[\![c?(y\,@w)\,P']\!] \;|\; \ell\langle\!\langle\Delta'\rangle\!\rangle$$

where $\Delta' = \Delta \sqcap \{m{:}\mathsf{loc}\{d{:}\mathsf{res}\langle\mathsf{bool}\rangle\}\}$. After the second communication, the network reduces to

$$\ell[\![P'']\!] \;|\; \ell\langle\!\langle\Delta''\rangle\!\rangle$$

where $\Delta'' = \Delta' \sqcap \{m{:}\mathsf{loc}\{d{:}\mathsf{res}\langle\mathsf{int}\rangle\}\} = \Delta \sqcap \{m{:}\mathsf{lbad}\}$. □

Table VI. Runtime Error

$$
\begin{array}{rll}
\ell[\![a?(X{:}\mathrm{T})\,P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle & \xrightarrow{err\,\ell} & \text{if } \Delta(\ell) \not<: \mathsf{loc}\{a{:}\mathsf{res}\langle\mathrm{T}\rangle\} \\
\ell[\![a!\langle v\rangle\,P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle & \xrightarrow{err\,\ell} & \text{if } \Delta(\ell) \not<: \mathsf{loc}\{a{:}\mathsf{res}\langle\mathrm{T}\rangle\},\ \text{ all T} \\
\ell[\![a!\langle v\rangle\,P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle & \xrightarrow{err\,\ell} & \text{if } \Delta(\ell) <: \mathsf{loc}\{a{:}\mathsf{res}\langle\mathrm{T}\rangle\} \text{ and } \Delta \sqcap \{_\ell v{:}\mathrm{T}\} \text{ undef} \\
\ell[\![\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] & \xrightarrow{err\,\ell} & \text{if } \{_\ell u{:}\mathrm{T}\} \text{ undef or } \{_\ell v{:}\mathrm{T}\} \text{ undef, all T}
\end{array}
$$

$$
\frac{N \xrightarrow{err\,\ell}}{(\nu_k\, e{:}\mathrm{T})\, N \xrightarrow{err\,\ell\{\!|^k/_e|\!\}}} \qquad \frac{N \xrightarrow{err\,\ell}}{N \mid M \xrightarrow{err\,\ell}} \qquad \frac{N \equiv M \quad M \xrightarrow{err\,\ell}}{N \xrightarrow{err\,\ell}}
$$

4.3. Subject Reduction and Type Safety

As we have seen in Section 3.2 partial typing is not preserved by the standard reduction relation. However this property is regained by the revised reduction relation of Table V. First we note that well-formedness (Definition 2) is preserved by reduction.

Proposition 3. *If P is well-formed and $P \longrightarrow P'$ then P' is well-formed.*

Proof. By composition of results for $\equiv$ and $\longmapsto$, which follow by a straightforward induction on judgements. □

Theorem 2 (Subject Reduction). *If $\Gamma \vdash N$ and $N \longrightarrow N'$ then $\Gamma \vdash N'$*

Proof. See Appendix A. □

A typing system is only of interest to the extent that it guarantees freedom from runtime errors. Here we describe the runtime errors captured by our system, which can be informally described as *misuse of resources at good sites.* Often the formulation of runtime errors is quite cumbersome as it involves the invention of a tagged version of the language, see [13, 21]. However in this case the presence of filters makes it straightforward.

In Table VI we define, for each location ℓ a unary predicate $\xrightarrow{err\,\ell}$ over networks. The judgement $N \xrightarrow{err\,\ell}$ should be read: "in the network N there is a runtime error at location ℓ". There are basically two kinds of errors which can occur. The first is an attempted use of a resource at a location ℓ when that resource has not been declared at that location. Filters have full local knowledge and therefore this error can occur if an agent attempts to use a resource at ℓ which does not appear in the filter at ℓ. This is formalised in the first two cases of the definition in Table VI.

The second kind of error occurs when there is a local inconsistency between values being manipulated by an agent. These may occur in either of two ways. The first, accounted for in the third clause in Table VI, is when a value is about to be transmitted locally which is inconsistent with the current contents of the filter. The second, accounted for in the fourth clause, is when the values in a match cannot be assigned the same type.

Finally, note that in the case that a location name m is restricted, errors at m are attributed to the site which created m (given as k in Table VI). This fact explains the need for the side condition $\mathrm{T} \neq \mathsf{lbad}$ on the rules ($\mathsf{thread\text{-}new_g}$) and ($\mathsf{net\text{-}new_g}$) in Table IV.

Theorem 3 (Type Safety). *If* $\Gamma \vdash N$ *and* $\Gamma(\ell) \neq \mathsf{lbad}$ *then* $N \overset{err\,\ell}{\not\longrightarrow}$

Proof. See Appendix A. □

It should be emphasised that Type Safety does not ensure that outputs to resources which have been declared at a given site will always be serviced; this property needs to be guaranteed by programming the site appropriately. But a more sophisticated type system has been developed in [4] to enforce such properties.

5. Trust

In the semantics of the last section all agents moving to a new site are dynamically typechecked before gaining entrance. In this section we consider an optimisation which allows for freer and more efficient movement across the network; in particular to alleviate the need for dynamic typechecking. The idea is to add *trust* between locations; a trusted site is guaranteed never to misbehave and therefore agents moving from a trusted site need not be dynamically typechecked.

Formally we introduce a new type constructor for *trusted location types*, $\mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{A}}\}$. The extended syntax of types is obtained by replacing the clause for location types with:

$$\mathrm{K} ::= \mathsf{lbad} \mid \mathsf{loc}\{\widetilde{a}{:}\widetilde{\mathrm{A}}, \widetilde{x}{:}\widetilde{\mathrm{B}}\} \mid \mathsf{ltrust}\{\widetilde{a}{:}\widetilde{\mathrm{A}}, \widetilde{x}{:}\widetilde{\mathrm{B}}\}$$

Note that (as with the addition of lbad) this extension increases the set of possible resource types. For example the type $\mathsf{res}\langle\mathsf{ltrust}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}\rangle$ is the type of a resource for communicating trusted locations which have an integer resource named a. Thus we may have trusted locations with certain resources for handling trusted data and others for handling untrusted data. In a similar vein we may have untrusted locations containing resources that communicate trusted data. As we shall see, these resources at untrusted locations cannot be used to increase the level of trust in a network.

The extension of the subtyping relation to these new types is based on two ideas:

- Every trusted location is also a location.
- Every trusted location guarantees good behaviour; therefore, a "bad" or untyped location can never be trusted by a good site. This means that the type lbad is no longer the minimal location type in the subtyping preorder.

The subtyping relation is therefore built up using the ordering:

$$\begin{tikzcd}[column sep=small]
 & & \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{A}}\} & \\
 & \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{A}}, v{:}\mathrm{B}\} \arrow[ur] & & \mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{A}}\} \arrow[ul] \\
\mathsf{lbad} \arrow[ur] & & \mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{A}}, v{:}\mathrm{B}\} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

The formal definition is given in Appendix A.

Proposition 4. *The set of types, extended with* lbad *and* ltrust*, under the subtyping preorder, has a partial meet operator.*

Proof. See Appendix A. □

With the addition of ltrust, the filters in a network may contain more detailed information about remote sites. Consider a network N which contains a filter $\ell\langle\!\langle\Delta\rangle\!\rangle$. As before, if k is not mentioned in Δ, this means that ℓ has no knowledge of k. But now there are three possibilities with respect to a remote location k mentioned in $\ell\langle\!\langle\Delta\rangle\!\rangle$:

- $\Delta(k)$ <: lbad, which means that ℓ has accumulated sufficient contradictory information about k to conclude that k is untyped.
- $\Delta(k)$ <: ltrust, which means that ℓ *trusts* k. Note that this notion of trust is asymmetric; ℓ may trust k without k trusting ℓ. Also note that in well-typed systems, the rule (net-filter) in Table V ensures that k, trusted by ℓ, cannot be an untyped location unless ℓ itself is untyped; this is enforced by the requirement that $\Gamma(k)$ <: $\Delta(k)$, since $\mathsf{lbad} \not<: \mathsf{ltrust}$.
- $\Delta(k)$ <: loc, which means that ℓ knows of k, but cannot determine whether or not k is well-typed.

As we have seen in the previous section, the information in a filter may increase as the network evolves, that is, $\ell\langle\!\langle\Delta\rangle\!\rangle$ may evolve to $\ell\langle\!\langle\Delta'\rangle\!\rangle$, where $\Delta' <: \Delta$. But the subtyping relation between types ensures that once a location k is deemed "bad" in $\ell\langle\!\langle\Delta\rangle\!\rangle$ it will remain so forever, and similarly with sites that are deemed "trusted". It is only the third category which may change. In 8 we have seen that new information may result in $\Delta(k)$ changing from loc to lbad. We shall soon see that new information can also "improve" the status of k from loc to ltrust.

With the addition of trust, we can revise the reduction relation of the previous section to eliminate dynamic typechecking of agents arriving from trusted sites. We adopt the semantics of Table V, replacing ($\mathsf{r_f}$-move) with:

$$(\mathsf{r_t\text{-}move})\ k[\![\mathsf{goto}\,\ell.\,P]\!]\ \mid\ \ell\langle\!\langle\Delta\rangle\!\rangle \longmapsto \ell[\![P]\!]\ \mid\ \ell\langle\!\langle\Delta\rangle\!\rangle \quad \text{if}\ \Delta(\ell) <: \mathsf{ltrust}\ \text{or}\ \Delta \Vdash_{\ell}^{k} P$$

Note that the presence of ltrust changes the importance of the condition $\mathsf{lbad} <: \mathrm{K}$ in the dynamic typing rule ($\mathsf{val_f\text{-}self_1}$). Whereas this condition was tautological in Section 4, here it is not. The side condition precludes the use of ($\mathsf{val_f\text{-}self_1}$) to infer $\Delta \Vdash_{\ell}^{k} k{:}\mathsf{ltrust}$. This is important, as it prevents bad sites from becoming trusted.

Example 9. *Let* $\Delta = \{\ell{:}\mathsf{loc}\{d{:}\mathsf{res}\langle\mathsf{ltrust}\rangle\}, k{:}\mathsf{ltrust}\}$ *and consider the network:*

$$\ell\langle\!\langle\Delta\rangle\!\rangle\ \mid\ \ell[\![d?(z)\,P]\!]\ \mid\ k[\![\mathsf{goto}\,\ell.\,d!\langle m\rangle]\!]\ \mid\ m[\![\mathsf{goto}\,\ell.\,d!\langle n\rangle]\!]$$

Here the locations m and n are unknown to ℓ, that is, $\Delta(m)$ and $\Delta(n)$ are undefined. In addition, d is a resource at ℓ for communicating trusted locations. The migration from m to ℓ is not immediately allowed since $\Delta \Vdash_{\ell}^{m} d!\langle n\rangle$ cannot be inferred; m does not have sufficient authority to convince ℓ that location n is to be trusted.

The move from k to ℓ, however, is *allowed, without dynamic typechecking, since ℓ trusts k. After the movement and communication on d, the resulting network is*

$$\ell\langle\!\langle\Delta'\rangle\!\rangle\ \mid\ \ell[\![P\{\!\!\{^{m}\!/\!_{z}\}\!\!\}]\!]\ \mid\ m[\![\mathsf{goto}\,\ell.\,d!\langle n\rangle]\!]$$

where $\Delta' = \Delta \sqcap \{m{:}\mathsf{ltrust}\}$. Thus, after communication with the agent from k, ℓ trusts m. At this stage the migration from m to ℓ is allowed, free of typechecking, and m can inform ℓ of another trusted site, n. In this way the web of trust *containing ℓ grows dynamically as the network evolves.*

Note it is crucial that ℓ trust k initially; if this were not the case then the original migration from k to ℓ would have been prevented by dynamic typechecking. There is no way for a site to "prove its trustworthiness"; the web of trust can only grow by communication between trusted sites.

□

Example 10. *Consider the network*

$$m[\![\mathsf{goto}\,\ell_0.\,\mathsf{goto}\,\ell_1.\,\mathsf{goto}\,\ell_2.\,P]\!] \;\mid\; \ell_i\langle\!\langle \Delta_i \rangle\!\rangle$$

where there is a web of trust among ℓ_i; that is $\Delta_i(\ell_j)$ <: ltrust for all i, j. Suppose further that $\Delta_0(m)$ is undefined, in particular that ℓ_0 does not trust m.

The migration from m to ℓ_0 is allowed only if the following judgement can be verified:

$$\Delta \Vdash^{m}_{\ell_0} \mathsf{goto}\,\ell_1.\,\mathsf{goto}\,\ell_1.\,\mathsf{goto}\,\ell_2.\,P$$

Note that this checks not only the potential behaviour of the incoming agent at the initial site ℓ_0 but also at the other sites ℓ_1, ℓ_2. So an agent is allowed into the web of trust between ℓ_0, ℓ_1 and ℓ_2 only if can be assured not to harm any resources at any of the locations in the web. Moreover this check is made against the knowledge at the incoming site ℓ_0. Even if P intends to respect all the resources at ℓ_2, if it mentions a resource at ℓ_2 of which Δ_0 is unaware, entry will be barred.

If the typecheck against Δ_0 succeeds then we obtain the network

$$\ell_0[\![\mathsf{goto}\,\ell_1.\,\mathsf{goto}\,\ell_2.\,P]\!] \;\mid\; \ell_i\langle\!\langle \Delta_i \rangle\!\rangle$$

where the agent from m has gained entry to the web of trust. The subsequent movements, from ℓ_0 to ℓ_1 and from ℓ_1 to ℓ_2, are allowed freely because of the relationship of trust between these sites. If P moves outside the web of trust, however, say to m, and then wishes to return to some ℓ_i, then it will be typechecked again before reentry. In Section 6.1, we give an example which shows that such typechecking is necessary for agents which wish to reenter a web of trust. □

Example 11. *As a final example, suppose that the set of locations is static and all sites are mutually trusted. In this case we recover the standard semantics (modulo the presence of filters), as given in Section 2.* □

The static typing relation remains unchanged from the previous section, although there is a certain redundancy of types in the static environments Γ. Since it is reasonable to suppose that sites trust themselves, we might wish to limit Γ to include only trusted and bad locations; however, none of our results require this.

The main results of the previous section extend to the new setting.

Theorem 4 (Subject Reduction). *If $\Gamma \vdash N$ and $N \longrightarrow N'$ then $\Gamma \vdash N'$*

Proof. See Appendix A. □

Theorem 5 (Type Safety). *If* $\Gamma \vdash N$ *and* $\Gamma(\ell) \neq \mathsf{lbad}$ *then* $N \overset{err\,\ell}{\not\longrightarrow}$

Proof. See Section A. □

6. Discussion

In this section we discuss some issues which arise in our formalisation of the semantics of open systems and point to some variations and extensions.

6.1. Authority

Note that as an agent moves about the network, it is always received at a site with the authority of the last location visited. Thus when $m[\![\mathsf{goto}\,k.\,\mathsf{goto}\,\ell.\,P]\!]$ arrives at ℓ, the thread P is typechecked under authority k, rather than m. An alternative would be to allow agents to maintain their authority as they move about the network. This alternative approach, however, is not compatible with our typing system. To see this, let us temporarily change the syntax of agents from $\ell[\![P]\!]$ to ${}^{k}_{\ell}[\![P]\!]$, meaning that the thread P is running at ℓ under the authority of k. Using this extended syntax, our move rule, from Table V can be expressed:

$$ {}^{k}_{m}[\![\mathsf{goto}\,\ell.\,P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle \longmapsto {}^{m}_{\ell}[\![P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle \quad \text{if } \Delta(m) <: \mathsf{ltrust} \text{ or } \Delta \Vdash^{m}_{\ell} P $$

The alternative semantics would be:

$$ {}^{k}_{m}[\![\mathsf{goto}\,\ell.\,P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle \longmapsto {}^{k}_{\ell}[\![P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle \quad \text{if } \Delta(k) <: \mathsf{ltrust} \text{ or } \Delta \Vdash^{k}_{\ell} P $$

Consider the following network, where $\mathrm{T} = \mathsf{res}\langle\mathsf{bool}\rangle@\mathsf{loc}$, typed using the environment Γ given in Section 3.2 (filters not shown):

$$
\begin{array}{ll}
& {}^{k}_{k}[\![\mathsf{goto}\,m.\,d?(x@z{:}\mathrm{T})\,\mathsf{goto}\,\ell.\,b!\langle x@z\rangle]\!] \mid {}^{m}_{m}[\![d!\langle a@k\rangle]\!] \\
\longmapsto & {}^{k}_{m}[\![d?(x@z{:}\mathrm{T})\,\mathsf{goto}\,\ell.\,b!\langle x@z\rangle]\!] \qquad\quad \mid {}^{m}_{m}[\![d!\langle a@k\rangle]\!] \\
\longmapsto & {}^{k}_{m}[\![\mathsf{goto}\,\ell.\,b!\langle a@k\rangle]\!] \\
\longmapsto & {}^{k}_{\ell}[\![b!\langle a@k\rangle]\!]
\end{array}
$$

All of these reductions are allowed by the alternative semantics, however, $\Gamma \nvdash {}^{k}_{\ell}[\![b!\langle a@k\rangle]\!]$. Since ℓ checks the incoming agent $b!\langle a@k\rangle$ under authority of k, it believes its assertion that a is a Boolean channel at k, whereas a is in fact an integer channel. This example formalises the intuition that agents can be polluted by visiting bad sites.

6.2. Filter Update

The reduction semantics in Table V includes certain mechanisms for updating filters. The rules ($\mathsf{r_f}$-newr) and ($\mathsf{r_f}$-newl) are necessary to ensure that restricted names are handled properly, in particular to ensure that well-formedness and well-typing are preserved by reduction. The rule ($\mathsf{r_f}$-comm), however, is just one of a number of possible ways in which filters can actively update their knowledge of remote sites. While ($\mathsf{r_f}$-comm) is simple and expressive, it may be expensive to implement. A more restricted approach would be to assign a special channel, say update, for which ($\mathsf{r_f}$-comm) applied, whereas all other channels would use the less expensive rule (r-comm), from the standard semantics. Another possibility would be to add analysis to the filter operation. Then the move rule would become:

$$k[\![\mathsf{goto}\,\ell.\,P]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle \longmapsto \ell[\![P]\!] \;\mid\; \ell\langle\!\langle\Delta \sqcap \Delta'\rangle\!\rangle \qquad \text{if } \Delta \Vdash^{k}_{\ell} P : \Delta'$$

The idea is that while checking an incoming term, the filter could also note any new names that are received with authority. Another possibility is to abandon non-local filter updates altogether; in this case, to allow a reasonable amount of expressiveness while preserving type safety, one would have to add further constructs to the language, as outlined at the end of the next subsection.

6.3. Progress

While subject reduction is important, it is purely a safety property; it does not imply that any reductions are ever performed. The semantics of Section 5 enjoys the property that whenever an agent attempts to move from a site k to a location that trusts k, the movement is always successful. This *liveness* property relies on the fact that the target trusts k, however. It works because agents from trusted sites come in with "universal authority", that is, the authority to say whatever they like.

A stronger property, which we call *progress*, is that whenever a well-typed agent attempts to move between two good locations, the movement is successful. Suppose we add the following clause to the definition of runtime error in Table VI:

$$k[\![\mathsf{goto}\,\ell.\,P]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle \xrightarrow{err\,\{k,\ell\}} \text{ if } k[\![\mathsf{goto}\,\ell.\,P]\!] \;\mid\; \ell\langle\!\langle\Delta\rangle\!\rangle \not\longrightarrow$$

We then say that the typing system guarantees *progress* if

$$\Gamma \vdash N \text{ and } \Gamma(\ell) \not<: \mathsf{lbad},\, \Gamma(k) \not<: \mathsf{lbad} \text{ implies } \neg(N \xrightarrow{err\,\{k,\ell\}})$$

Note that this property is not dependent on the trust relation between sites. Unfortunately, this progress property does not hold for our semantics, as can be seen from the following example. Let Γ, Δ and N be defined as follows:

$$\Gamma = \left\{ \begin{array}{l} k : \mathsf{ltrust}\{a : \mathsf{res}\langle\mathsf{int}\rangle\} \\ \ell : \mathsf{ltrust}\{c : \mathsf{res}\langle\mathsf{res}\langle\mathsf{int}\rangle@\mathsf{loc}\rangle\} \\ m : \mathsf{ltrust} \end{array} \right\}$$

$$\Delta = \{ \ \ \ell : \mathsf{ltrust}\{b : \mathsf{res}\langle\mathsf{res}\langle\mathsf{bool}\rangle@\mathsf{loc}\rangle\} \ \}$$

$$N = m[\![\mathsf{goto}\, \ell.\ c!\langle a@k\rangle]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle$$

Then $\Gamma \vdash N$, but $N \not\rightarrow$. The problem here is that, although the agent at m is well-typed, the reference to a is made without authority.

In practise, progress may not be that important, depending upon the application and the underlying implementation. In the example above where the move from m to ℓ is unsuccessful, an implementation of the filter at ℓ might report to m the reasons for the failure. It would then be up to m to resend the agent (or some piece of it) via k.

On the other hand, one way to guarantee progress would be to allow an incoming agent to refer only to *local* values or values at its *authority*. It is straightforward to design a static type system to enforce this constraint.[4] However such an approach is very restrictive without some addition to the language. One possibility would be to introduce the notion of *signed* values (possibly based on [1]) which would allow certain values in an agent to be received (and typed) under a *different authority* than that of the agent itself. Even without full progress, signatures could be useful. In the example sketched above, after m's agent is refused entry to ℓ, m might itself resend the agent, rather than forwarding it to k, this time carrying a signed value to prove that $a@k$ is of the appropriate type.

6.4. Anonymous Networks

In [12] we presented a semantics for open system in which the authority of incoming agents is not known. We call such systems *anonymous networks*. In Section B we recast the semantics of [12] using filters and lbad. An attractive property of the semantics is that filter updating is purely local, that is, no non-local data need be stored in filters.

[4] One possibility is to change the move rule to read:

$$\frac{\{w{:}\Gamma(w), u{:}\Gamma(u)\} \vdash_u P}{\Gamma \vdash_w \mathsf{goto}\, u.\, P}$$

However because the origin of incoming agents cannot be determined it is not possible to incorporate notions of trust into this semantics, which implies that incoming agents must always be typechecked. In addition, it is very easy for good sites to develop misconceptions about other good sites, frustrating progress.

7. Conclusions

We introduced the notion of *partial typing*, which captures the intuition that "bad" sites in a network may harbour malicious agents while "good" sites may not. We demonstrated that in the presence of partial typing, some form of dynamic typechecking is required to ensure that good sites remain uncorrupted. We presented a semantics for Dπ incorporating such dynamic typechecking, showing that it prevented type violations at good sites, and discussed the extent to which it guaranteed progress. Finally, we added *webs of trust* to the language, reducing the need for dynamic typechecking while retaining type safety at good sites.

The presentation of Dπ given here differs somewhat from that of [13]; for example, we have added base types and moved some of the semantic rules from the structural equivalence to the reduction relation. Most of the changes are stylistic rather than substantive. Two of the changes, however, are essential for the treatment of partial typing. First, we have moved the rule (r-new) from the structural equivalence to the reduction relation; this is necessary to allow filter updating. Second, we have split the space of names in two, syntactically distinguishing locations from resources; this is necessary to prevent the filter updating rules from producing nonsense environments such as $\{\ell{:}\mathsf{loc}\{\ell{:}\mathsf{res}\langle\rangle\}\}$.

Several other distributed variants of the π-calculus have been defined, and it is informative to see how partial typing might be added to these languages. Syntactically, Dπ is most similar to the language of Amadio and Prasad [3, 2], which also uses a "goto" operator for thread movement, written "$\mathsf{spawn}(\ell, P)$". However, in Amadio and Prasad's language, the set of resources available to a thread does not vary as the thread moves about the network. This means that an agent at ℓ can access resources at a different location k without requiring thread movement. While this makes the language very expressive, it also frustrates the use of filters to typecheck incoming threads. To add partial typing to such a language, one would need to typecheck *messages* dynamically, rather than threads, violating the third principle given in the introduction. In addition, the fact that names are assigned unique locations in [2] appears to be incompatible with partial typing, as outlined at the end of 4.1.

The join calculus of Fournet, Gonthier, Levy, Marganget and Remy [10] shares many of these properties. Whereas Amadio's language adds thread movement to message movement, however, the join calculus adds location movement. Unfortunately this does not help combat the problems outlined above, which result from the "universal extent" of resource names in both subject and object position. In Dπ, the type system ensures that the "extent" of resource names in subject position is local, that is, resources may be *referenced* at remote sites, but may only be *used* locally.

Klaim, [22], is another location-aware distributed language, where data and processes may be transferred between sites, although there is a differentiation between logical locations and physical sites. Computations are based on local *tuple spaces*, which support the co-ordination operations of *read*, *write* and *execute*. There is a well-developed type system, [20], which views these operations as capabilities, and which can be used to control access to local resources; this is similar in style to the work in [13]. However a recent technical report, [11], has considered a subset of Klaim, called μ-Klaim, and developed a combination of static and dynamic typing which allows locations to dynamically acquire local rights to perform the individual co-ordination operations.

Cardelli and Gordon's ambient calculus [7], also appears to be amenable to partial typing since ambient movement is a local operation; thus the problem of "universal extent" does not arise. The typing system of Dπ is based on the original sorting system of the π-calculus [15], and this system has been extended to the ambient calculus [6]. Whereas locations in Dπ have a straightforward analog in implementations — they correspond to address spaces — the notion of "ambient" is more general, adding expressiveness while blurring the distinction between agent movement and agent interaction. In the ambient calculus it is the open operator, rather than the in or out operators, which enables interaction between two threads (or thread collections). Thus a first attempt at partial typing for the ambient calculus would dynamically typecheck thread collections whenever they are opened. Since each ambient has only one "resource" (λ), however, this implies that dynamic typechecking must occur before every interaction, again violating our third principle. To get around this, one might introduce a type system for ambients which distinguished two types of ambients: those which typecheck incoming ambients and those which do not. The former would be similar to our locations, the latter, our resources. This discipline would open the possibility of typing code during in and out operations, rather than open. However there are now a large number of variants on the basic idea of ambients, together with a large number of different typing systems, many of which are proposing alternatives to the open operation; see [5, 8] for typical examples. Thus it remains to be seen which will provide the most convenient framework for dynamic typechecking.

We view our approach to guaranteeing safety type-properties as being a variation to the general concept of *proof carrying code* (PCC), [19]. There (untrusted) code producers are expected to provide proof that their code satisfies some publicly available security policy. In [19] policies are expressed in first-order predicate calculus, together with axioms and inference rules which code producers may use to construct their proofs. In [9, 17] the approach is more constrained; types are used to express safety properties and to annotate assembly language instructions with type information, indicating the intention of the instructions. Thus typechecking plays the role of checking the validity of proofs. This is quite close, in principle, to the ideas outlined in this paper, although we have worked at the source-code level. We have also tackled issues more concerned with distributed computation, in particular how to handle third-party information about third-party resources.

Acknowledgements

The ideas presented in the paper have been sharpened by discussion with Alan Jeffrey and by questions from audiences at NCSU and De Paul, where preliminary versions of this work were presented. An extended abstract based on this paper was originally reported in [23]. The authors would also like to thank the anonymous referees for their insightful comments.

Appendix

A. Proofs

The Subject Reduction and Type Safety results for Section 4 are special cases of those of Section 5, in which no trusted types appear. We present only the more general results. First we establish Proposition 4.

The formal definition of subtyping with lbad and ltrust is:

$$\begin{array}{rr} \mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}},\ \widetilde{v}{:}\widetilde{\mathrm{T}}\} <: \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} & \mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}},\ \widetilde{v}{:}\widetilde{\mathrm{T}}\} <: \mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \\ \mathsf{lbad} <: \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} & \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}},\ \widetilde{v}{:}\widetilde{\mathrm{T}}\} <: \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \\ & \mathsf{lbad} <: \mathsf{lbad} \end{array}$$

Proposition 1 (Proposition 4). *The set of types, extended with* lbad *and* ltrust*, under the subtyping preorder, has a partial meet operator.*

Proof. Ignoring resources, the meet operator can be defined as follows:

	lbad	loc	ltrust
lbad	lbad	lbad	*undef*
loc	lbad	loc	ltrust
ltrust	*undef*	ltrust	ltrust

Combining this with the subtyping rules already given for resources, we have (omitting symmetric cases):

$$\begin{aligned}
\mathsf{lbad} \sqcap \mathsf{lbad} &= \mathsf{lbad}\\
\mathsf{lbad} \sqcap \mathsf{loc}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} &= \mathsf{lbad}\\
\mathsf{lbad} \sqcap \mathsf{ltrust}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} &= \mathit{undefined}\\
\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \sqcap \mathsf{loc}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} &= \begin{cases}\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}} \cup \widetilde{v}{:}\widetilde{\mathrm{T}}\} & \text{if } \forall i,j : u_i = v_j \text{ implies } \mathrm{S}_i = \mathrm{T}_j\\ \mathsf{lbad} & \text{otherwise}\end{cases}\\
\mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \sqcap \mathsf{ltrust}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} &= \begin{cases}\mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}} \cup \widetilde{v}{:}\widetilde{\mathrm{T}}\} & \text{if } \forall i,j : u_i = v_j \text{ implies } \mathrm{S}_i = \mathrm{T}_j\\ \mathit{undefined} & \text{otherwise}\end{cases}\\
\mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \sqcap \mathsf{ltrust}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} &= \begin{cases}\mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}} \cup \widetilde{v}{:}\widetilde{\mathrm{T}}\} & \text{if } \forall i,j : u_i = v_j \text{ implies } \mathrm{S}_i = \mathrm{T}_j\\ \mathit{undefined} & \text{otherwise}\end{cases}
\end{aligned}$$

The proof that this definition meets the requirements of Definition 1 follows by straightforward calculations, with a rather tedious case analysis for each result. □

The proofs of Subject Reduction and Type Safety use the fact that 2 extends to the type system with lbad and ltrust.

Theorem 1 (Theorem 4). *If* $\Gamma \vdash N$ *and* $N \longrightarrow N'$ *then* $\Gamma \vdash N'$

Proof. The result follows from results for the structural congruence and reduction precongruence:

$$\begin{array}{l}
\text{If } N \equiv N' \text{ then } \Gamma \vdash N \text{ if and only if } \Gamma \vdash N'\\
\text{If } \Gamma \vdash N \text{ and } N \longmapsto N' \text{ then } \Gamma \vdash N'
\end{array}$$

The first result is proved by induction on the definition of $\equiv$, the second by induction on the definition of $\longmapsto$. The proofs of both results, and the accompanying lemmas, can easily be derived from those found in [13]; in particular see Lemmas 4.7 and A.2, Proposition 4.5 and Theorem 5.1 of that paper. The only substantial differences are in the rules ($\mathsf{r_f}$-comm) and ($\mathsf{r_f}$-move), which we discuss below.

For the most part, the proof for ($\mathsf{r_f}$-comm) follows that given in [13]. The only additional complication is presence of filter updating. Suppose that $\Gamma(k) \neq \mathsf{lbad}$, $\Gamma \vdash_k v{:}\mathrm{T}$ and Γ <: Δ. We must show that $\Delta' =$

$\Delta \sqcap \{_k v{:}\mathrm{T}\}$ is defined and that $\Gamma <: \Delta'$, but this follows immediately from a and a.

Now let us turn to ($\mathsf{r_f}$-move). Suppose that $\Gamma \vdash k[\![\mathsf{goto}\,\ell.\,P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle$ and $k[\![\mathsf{goto}\,\ell.\,P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle \longmapsto \ell[\![P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle$. To establish the result, it is sufficient to show that $\Gamma \vdash_{\ell} P$. There are three cases to consider:

- Suppose $\Gamma(\ell) = \mathsf{lbad}$. The result follows from ($\mathsf{thread\text{-}bad}$).
- Suppose that $\Gamma(\ell) \neq \mathsf{lbad}$ and $\Gamma(k) \neq \mathsf{lbad}$. The result follows from $\Gamma \vdash k[\![\mathsf{goto}\,\ell.\,P]\!]$, using ($\mathsf{thread\text{-}move}$).
- Suppose that $\Gamma(\ell) \neq \mathsf{lbad}$ and $\Gamma(k) = \mathsf{lbad}$. Since ($\mathsf{net\text{-}filter_g}$) requires $\Gamma <: \Delta$ and $\mathsf{lbad} \not<: \mathsf{ltrust}$, it cannot be that $\Delta(k) <: \mathsf{ltrust}$. Therefore in order for reduction to occur it must be that $\Delta \Vdash_{\ell}^{k} P$. But using the proof that $\Delta \Vdash_{\ell}^{k} P$, we can construct a proof that $\Gamma \vdash_{\ell} P$. Most of the rules, in fact, are identical. The only difficulty is to establish the validity of the addition rules for dynamic typing given in Table V. In these cases, we proceed as follows:

  ($\mathsf{val_f\text{-}self_1}$) Let K be a location type such that $\mathsf{lbad} <: \mathrm{K}$. Then $\Gamma \vdash_w k{:}\mathrm{K}$, as required.

  ($\mathsf{val_f\text{-}self_2}$) For any a and A, $\mathsf{lbad} <: \mathsf{loc}\{a{:}\mathrm{A}\}$; thus $\Gamma \vdash_k a{:}\mathrm{A}$.

  ($\mathsf{val_f\text{-}return}$) By ($\mathsf{thread\text{-}bad}$), $\Gamma \vdash_k P$; therefore $\Gamma \vdash_w \mathsf{goto}\,k.\,P$. $\Box$

Theorem 2 (Theorem 5). *if* $\Gamma \vdash N$ *and* $\Gamma(\ell) \neq \mathsf{lbad}$ *then* $N \overset{err\,\ell}{\not\longrightarrow}$

Proof. We prove the contrapositive, that is, that $N \xrightarrow{err\,\ell}$ and $\Gamma(\ell) \neq \mathsf{lbad}$ imply $\Gamma \nvdash N$. The proof proceeds by induction on the derivation of $N \xrightarrow{err\,\ell}$. We present four representative cases:

- Suppose that $\ell[\![a!\langle v\rangle\, P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle \xrightarrow{err\,\ell}$ because for all T, $\Delta(\ell) \not<: \mathsf{loc}\{a{:}\mathsf{res}\langle\mathrm{T}\rangle\}$. Since $\Gamma(\ell) \neq \mathsf{lbad}$, we have $\Gamma(\ell) = \Delta(\ell)$, so clearly $\Gamma \nvdash_{\ell} a{:}\mathsf{res}\langle\mathrm{T}\rangle$. Thus we have that $\Gamma \nvdash \ell[\![a!\langle v\rangle\, P]\!]$, as required.
- Suppose that $\ell[\![a!\langle v\rangle\, P]\!] \mid \ell\langle\!\langle\Delta\rangle\!\rangle \xrightarrow{err\,\ell}$ because $\Delta(\ell) <: \mathsf{loc}\{a{:}\mathsf{res}\langle\mathrm{T}\rangle\}$ and $\Delta \sqcap \{_\ell v{:}\mathrm{T}\}$ is undefined. By way of contradiction, suppose further that $\Gamma \vdash_{\ell} v{:}\mathrm{T}$. Using Lemma 2(a) and Lemma 2(a), we have that $\Delta \sqcap \{_\ell v{:}\mathrm{T}\}$ is defined, a contradiction. Thus it must be that $\Gamma \nvdash \ell[\![a!\langle v\rangle\, P]\!]$.
- Suppose that $\ell[\![\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \xrightarrow{err\,\ell}$ because for every R either $\{_\ell u{:}\mathrm{R}\}$ or $\{_\ell v{:}\mathrm{R}\}$ is undefined. By way of contradiction, suppose that $\Gamma \vdash_{\ell} \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$ and therefore for some S, T we have:

$$\Gamma \vdash_{\ell} u{:}\mathrm{S} \qquad \Gamma \vdash_{\ell} v{:}\mathrm{T} \qquad \Gamma \sqcap \{_\ell u{:}\mathrm{T}\} \sqcap \{_\ell v{:}\mathrm{S}\}\ \text{defined}$$

By Lemma 2(a), therefore, $\{_\ell u{:}\mathrm{S}\}$ defined and $\Gamma <: \{_\ell u{:}\mathrm{S}\}$. Hence, by Lemma 2(b) we have that $\{_\ell u{:}\mathrm{S}\}\sqcap\{_\ell u{:}\mathrm{T}\}$ defined. Finally, using Lemma 2(b) we have that $\{_\ell u{:}(\mathrm{S}\sqcap\mathrm{T})\}$ defined. Let $\mathrm{R} = \mathrm{S}\sqcap\mathrm{T}$. Symmetrically, we can conclude that $\{_\ell v{:}\mathrm{R}\}$ is also defined, leading to a contradiction.

– Finally, suppose that $(\nu_\ell k{:}\mathrm{K})\,N \xrightarrow{err\,\ell}$ because $N \xrightarrow{err\,k}$. If $\Gamma \vdash (\nu_\ell k{:}\mathrm{K})\,N$ then (since $\Gamma(\ell) \neq \mathsf{lbad}$) we have from ($\mathsf{net\text{-}new_g}$) that $\mathrm{K} \neq \mathsf{lbad}$, thus we can apply induction to conclude that $N \not\xrightarrow{err\,k}$, a contradiction. □

B. Anonymous Networks

In this section we describe how the techniques developed in this paper could be brought to bear on the "anonymous" networks of [12]. As a starting place, we take the standard semantics of Section 2 under the partial typing relation of Section 3.

As in Section 4, we extend the syntax of networks to include filters, although here they are of the form $k\langle\!\langle \mathrm{K}\rangle\!\rangle$, rather than $k\langle\!\langle\Delta\rangle\!\rangle$. Filters need record only information about *local* resources. The typing rules for filters are:

$$(\mathsf{net_a\text{-}filter_g})\frac{\Gamma(k) = \mathrm{K}}{\Gamma \vdash k\langle\!\langle \mathrm{K}\rangle\!\rangle} \qquad (\mathsf{net_a\text{-}filter_b})\frac{\Gamma(k) = \mathsf{lbad}}{\Gamma \vdash k\langle\!\langle \mathrm{K}\rangle\!\rangle}$$

The reduction rules are as in Section 2.2, but for ($\mathsf{r\text{-}move}$) and ($\mathsf{r\text{-}new}$), which become:

$$\begin{array}{llll}
(\mathsf{r_a\text{-}move}) & \ell\langle\!\langle \mathrm{L}\rangle\!\rangle \mid k[\![\mathsf{goto}\,\ell.\,P]\!] & \longmapsto \ell\langle\!\langle \mathrm{L}\rangle\!\rangle \mid \ell[\![P]\!] & \text{if } k=\ell \text{ or } \{\ell{:}\mathrm{L}\} \vdash_{\ell} P \\
(\mathsf{r_a\text{-}newr}) & k\langle\!\langle \mathrm{K}\rangle\!\rangle \mid k[\![(\nu a{:}\mathrm{A})\,P]\!] & \longmapsto (\nu_k\, a{:}\mathrm{A})(k[\![P]\!] \mid k\langle\!\langle \mathrm{K}\sqcap \mathsf{loc}\{a{:}\mathrm{A}\}\rangle\!\rangle) & \text{if } a \notin \mathrm{fn}(\mathrm{K}) \\
(\mathsf{r_a\text{-}newl}) & k[\![(\nu\ell{:}\mathrm{L})\,P]\!] & \longmapsto (\nu_k \ell{:}\mathrm{L})(k[\![P]\!] \mid \ell\langle\!\langle \mathrm{L}\rangle\!\rangle) & \text{if } \ell \neq k
\end{array}$$

The static and dynamic typing relations are the same. The rules are the same as those given in Table IV, but for the rule for agents, which becomes:

$$(\mathsf{net_a\text{-}agent})\frac{\{k{:}\Gamma(k)\} \vdash_{k} P}{\Gamma \vdash k[\![P]\!]}$$

In addition, we add the following three rules, which correspond to the three rules for dynamic typing added in Figure V:

$$(\mathsf{val_a\text{-}self_1})\frac{k \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{w} k{:}\mathrm{K}} \quad (\mathsf{val_a\text{-}self_2})\frac{k \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{k} a{:}\mathrm{A}} \quad (\mathsf{thread_a\text{-}remote})\frac{k \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{w} \mathsf{goto}\,k.\,P}$$

Note that the definition of static typing here is much weaker than that presented in the body of the paper. For example the network alth(6) of Section 3.2 is well-typed, although (8) is not. Using these definitions, one can establish Subject Reduction and a weaker notion of Type Safety (given in [12]).

This formulation has certain advantages over that of [12], such as the stronger language of partial types. Moreover it allows self moves to go untyped; that is, reductions of the form $\ell[\![\mathsf{goto}\,\ell.\,P]\!] \longmapsto \ell[\![P]\!]$ are always allowed.

References

1. Abadi, M. and A. Gordon: 1999, 'A Calculus for Cryptographic Protocols: The Spi Calculus'. *Information and Computation* **148**, 1–70.
2. Amadio, R.: 1997, 'An asynchronous model of locality, failure, and process mobility'. In: *COORDINATION '97*, Vol. 1282 of *Lecture Notes in Computer Science*.
3. Amadio, R. and S. Prasad: 1994, 'Localities and Failures'. In: *Proc. 14th Foundations of Software Technology and Theoretical Computer Science*, Vol. 880 of *Lecture Notes in Computer Science*.
4. Boudol, G., R. Amadio, and C. Lhoussaine: 1999, 'The receptive distributed pi-calculus'. In: *Proceedings of the FST-TCS'99*. pp. 304–315.
5. Bugliesi, M., S. Crafa, M. Merro, and V. Sassone: 2002, 'Communication Interference in Mobile Boxed Ambients'. In: *FSTTCS'02: Int. Conf. on Foundations of Software Technology and Theoretical Computer Science*.
6. Cardelli, L. and A. D. Gordon: 1999, 'Types for Mobile Ambients'. In: *Symposium on Principles of Programming Languages*. pp. 79–92.
7. Cardelli, L. and A. D. Gordon: 2000, 'Mobile Ambients'. *Theoretical Computer Science* **240**(1), 177–213. An extended abstract appeared in *Proceedings of FoSSaCS '98*: 140–155.
8. Coppo, M., M. Dezani-Ciancaglini, E. Giovannetti, and I. Salvo: 2003, 'M3: Mobility types for mobile processes in mobile ambients'. In: *Electronic Notes in Theoretical Computer Science*, Vol. 70.
9. Crary, K., D. Walker, and G. Morrisett: 1999, 'Typed Memory Management in a Calculus of Capabilities'. In: *In Twenty-Sixth ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, Texas*. pp. 262–275.
10. Fournet, C., G. Gonthier, J. Levy, L. Marganget, and D. Remy: 1996, 'A Calculus of Mobile Agents'. In: U. Montanari and V. Sassone (eds.): *CONCUR: Proceedings of the International Conference on Concurrency Theory*, Vol. 1119 of *Lecture Notes in Computer Science*. Pisa, pp. 406–421.
11. Gorla, D. and R. pugliese: 2003, 'Resource Access Control and Dynamic Privileges Acquision'. Technical report, University of Florence.
12. Hennessy, M. and J. Riely: 1998, 'Type-Safe Execution of Mobile Agents in Anonymous Networks'. Computer Science Technical Report 3/98, University of Sussex. Available from `http://www.informatics.sussex.ac.uk/`.
13. Hennessy, M. and J. Riely: 2002, 'Resource access control in systems of mobile agents'. *Information and Computation* **173**, 82–120.

14. Kozen, D.: 1988, 'Efficient Code Certification'. Technical Report 98-1661, Cornell University, Department of Computer Science. Available from `http://www.cs.cornell.edu/kozen/secure`.
15. Milner, R.: 1991, '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. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.
16. Milner, R., J. Parrow, and D. Walker: 1992, 'A Calculus of Mobile Processes, Parts I and II'. *Information and Computation* **100**, 1–77.
17. Morrisett, G., K. Crary, N. Glew, D. Grossman, R. Samuels, F. Smith, D. Walker, S. Weirich, and S. Zdancewic: 1999, 'TALx86: A Realistic Typed Assembly Language'. In: *In the 1999 ACM SIGPLAN Workshop on Compiler Support for System Software, Atlanta*. pp. 25–35.
18. Morrisett, G., D. Walker, K. Crary, and N. Glew: 1998, 'From System F to Typed Assembly Language'. In: *Conference Record of the ACM Symposium on Principles of Programming Languages*. San Diego, pp. 85–97.
19. Necula, G. C.: 1997, 'Proof-Carrying Code'. In: *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Paris, France, pp. 106–119.
20. Nicola, R. D., G. Ferrari, R. Pugliese, and B. Venneri: 2000, 'Types for Access Control'. *Theoretical Computer Science* **240**(1), 215–254.
21. Pierce, B. and D. Sangiorgi: 1996, 'Typing and Subtyping for Mobile Processes'. *Mathematical Structures in Computer Science* **6**(5), 409–454. Extended abstract in LICS '93.
22. R. DeNicola, G. Ferrari, R. P.: 1998, 'Klaim: a Kernel Language for Agents Interaction and Mobility'. In: *IEEE Transactions on Software Engineering*. pp. 315–330.
23. Riely, J. and M. Hennessy: 1999, 'Trust and Partial Typing in Open Systems of Mobile Agents (extended abstract)'. In: *Conference Record of POPL '99 The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. pp. 93–104.
24. Sewell, P.: 1997, 'Global/Local Subtyping for a Distributed π-calculus'. Technical Report 435, Computer Laboratory, University of Cambridge.
25. Yellin, F.: 1995, 'Low-Level Security in Java'. In: *WWW4 Conference*. Available from `http://www.javasoft.com/sfaq/verifier.html`.