Trust and Partial Typing in Open Systems of Mobile Agents

(Extended Abstract)

James Riely[*] Matthew Hennessy[†]

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 [12] we presented a type system for controlling the use of resources in a distributed system, or network. In particular it guarantees that resource access is always safe, *e.g.* integer resources are always accessed with integers and string resources are always accessed with strings. While this property is desirable, it is a property of the network as a whole. In open systems it is impossible to verify the system as a whole, *e.g.* to "type-check the web". In this paper, we present type systems and semantics which guarantee safe resource access for open systems in which some sites are untyped.

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 [2]. In such languages code signatures and nonces are directly manipulable as program objects. Here we take a more abstract approach, presenting a "secure" semantics 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.

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 harbor malicious agents.
2. Malicious agents should not be able to corrupt computation at good sites; however, not all agents at bad sites are 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 type-checking.

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 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, a distributed variant of the π-calculus [17]. The calculus was introduced in [23], but here we use the more recent formulation given in [12]. 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

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

is the type of a location with four resources, two for manipulating integers and two for manipulating location names. A feature which distinguishes Dπ from related languages [10, 4, 25] is that resource names have only local significance, *i.e.* resource names are unique locally, but not globally across the network.

To formalize 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

[*] http://www.csc.ncsu.edu/eos/users/r/riely/www/. Department of Computer Sciencess, North Carolina State University, Raleigh, NC 27695-7534 USA. riely@csc.ncsu.edu. Research funded by NSF grant EIA-9805604.

[†] http://www.cogs.susx.ac.uk/users/matthewh/. School of Cognitive and Computing Sciences, Univ. of Sussex, Falmer, Brighton, BN1 9QH UK. matthewh@cogs.susx.ac.uk. Research funded by CONFER II and EPSRC project GR/K60701.

To appear in the 26th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, January 1999, San Antonio, Texas. Full version available as Computer Science Technical Report 98/04, University of Sussex, http://www.cogs.susx.ac.uk/.

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 Section 3.2. Technically this means that partial typing is not preserved by the standard reduction semantics of Dπ; a good site will cease to be well-typed if an untypable agent moves there from an untyped site. The object of this paper is to formalize a protection policy for good sites against such malicious attacks.

As in [27, 19, 15, 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. But 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. 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 [13], 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 a site to another. To alleviate this burden, in Section 5 we introduce a relationship of *trust* between locations, formalized 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.

The paper proceeds as follows. In Section 2 we review Dπ and its standard semantics, including the standard typing system. Section 3 introduces the notion of partial typing and shows that partial typing is not preserved by the standard reduction relation. Section 4 presents the formalization of filters and dynamic typing, showing how these are incorporated into the run-time semantics. In Section 5 this framework is extended to include trust. The paper ends with a brief discussion of related work.

In this extended abstract all proofs are omitted, as is much of the discussion. The full version [22] is available at `ftp://ftp.cogs.susx.ac.uk/pub/reports/compsci/cs0498.ps.Z`.

Table 1 Syntax

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

2 The Language and Standard Typing

In this section we review the syntax and standard semantics of Dπ, extended with base values. For a full treatment of the language, including many examples, see [12].

2.1 Syntax

The syntax is given in Table 1, although discussion of types, T, is postponed to Section 2.3. The syntax is parameterized 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–m,
- *Res*, of *resource names*, ranged over by a–d,
- *Var*, of *variables*, ranged over by x–z.

The main syntactic categories of the language are as follows:

- *Threads*, P–R, are terms of the ordinary polyadic π-calculus [16] with additional constructs for agent 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.

These, in turn use *Names* e, which include location names and resource names, *Values*, u–w, which include base values, names, variables and tuples of values and *Patterns*, X–Y, which include variables and tuples of patterns; we require that patterns be linear, *i.e.* that each variable appear at most once.

As an example of a network, consider the term:

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

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.

In the sequel we use $\mathrm{fv}(N)$, respectively $\mathrm{fn}(N)$, to denote the set of variables, respectively names, that occur free in N. A term with no free variables is *closed*. We write $P\{\!|{}^{u}\!/\!_{X}|\!\}$ for the capture-avoiding substitution of u for X in P. We adopt standard abbreviations from the π-calculus [17], *e.g.* dropping final occurrences of stop, writing $u_1, .., u_n$ as $\widetilde{u}$, and identifying terms up to renaming of bound names and variables. We also systematically omit type annotations when they play no role in the discussion, for example rendering $u?(X{:}\mathrm{T})\,P$ as $u?(X)\,P$.

2.2 Standard Reduction

The standard reduction semantics is defined using two relations over closed network terms, a reduction relation ($M \longmapsto N$) and a structural equivalence ($M \equiv N$). The relations are defined using a set of axioms. So that these axioms may be applied anywhere within a network context, we introduce the idea of a network pre-congruence. A relation $\succ$ over networks is defined to be a *network pre-congruence* if $N \succ N'$ implies

- $N \,|\, M \succ N' \,|\, M$,
- $M \,|\, N \succ M \,|\, N'$, and
- $(\nu_k e{:}\mathrm{T})\,N \succ (\nu_k e{:}\mathrm{T})\,N'$.

The reduction pre-congruence is formalized as the least network pre-congruence which satisfies the axioms given in Table 2. The reduction axioms for communication ($\mathsf{r\text{-}comm}$) and matching ($\mathsf{r\text{-}eq}$) 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 colocated agents. The most important new rule is ($\mathsf{r\text{-}move}$), $k\llbracket \mathsf{go}\,\ell.P\rrbracket \longmapsto \ell\llbracket P\rrbracket$, which states that an agent located at k can move to ℓ using the move operator $\mathsf{go}\,\ell.P$. Also significant is ($\mathsf{r\text{-}new}$), $k\llbracket(\nu e{:}\mathrm{T})\,P\rrbracket \longmapsto (\nu_k e{:}\mathrm{T})\,k\llbracket P\rrbracket$, 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 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 ($\mathsf{r\text{-}split}$), $k\llbracket P \,|\, Q\rrbracket \longmapsto k\llbracket P\rrbracket \,|\, k\llbracket Q\rrbracket$, allows an agent to spawn off subagents 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 ($\mathsf{r\text{-}new}$).

The structural congruence is defined similarly; it is the least network pre-congruence that satisfies the axioms of Table 2, together with the laws for equivalences and the standard monoid laws for parallel composition.[1] The axioms given in Table 2 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 main reduction relation we are interested in is $(\longrightarrow) = (\equiv \cdot \longmapsto \cdot \equiv)$; this allows structurally congruent networks to be considered "the same" from the point of view of reductions. As an example, suppose that we wish to write a network with two agents, one at k and one at ℓ. The agent at k wishes to send a fresh integer channel a, located at k, to the other agent using the channel c, located at ℓ. This network could be written:

$$\begin{array}{lll}
 & \ell\llbracket c?(z,x)\,Q\rrbracket \mid k\llbracket(\nu a)\ (P \mid \mathsf{go}\,\ell.\,c!\langle k,a\rangle)\rrbracket & \\
\longrightarrow & \ell\llbracket c?(z,x)\,Q\rrbracket \mid (\nu_k a)\left(k\llbracket P \mid \mathsf{go}\,\ell.\,c!\langle k,a\rangle\rrbracket\right) & (\mathsf{r\text{-}new}) \\
\longrightarrow & \ell\llbracket c?(z,x)\,Q\rrbracket \mid (\nu_k a)\left(k\llbracket P\rrbracket \mid k\llbracket\mathsf{go}\,\ell.\,c!\langle k,a\rangle\rrbracket\right) & (\mathsf{r\text{-}split}) \\
\longrightarrow & \ell\llbracket c?(z,x)\,Q\rrbracket \mid (\nu_k a)\left(k\llbracket P\rrbracket \mid \ell\llbracket c!\langle k,a\rangle\rrbracket\right) & (\mathsf{r\text{-}move}) \\
\longrightarrow & (\nu_k a)\,\ell\llbracket Q\{\!|{}^{k,a}\!/\!_{z,x}|\!\}\rrbracket \mid k\llbracket P\rrbracket & (\mathsf{s\text{-}extr})\,(\mathsf{r\text{-}comm})\,(\mathsf{s\text{-}grbg_2})
\end{array}$$

[1] The equivalence laws are: $M \equiv M$, $M \equiv N$ implies $N \equiv M$, and $M \equiv N$ and $N \equiv O$ imply $M \equiv O$. The monoid laws are: $M \,|\, \mathbf{0} \equiv M$, $M \,|\, N \equiv N \,|\, M$, and $M \,|\, (N \,|\, O) \equiv (M \,|\, N) \,|\, O$.

Table 2 Standard Reduction

Reduction pre-congruence:

($\mathsf{r\text{-}move}$)	$k\llbracket\mathsf{go}\,\ell.P\rrbracket$	$\longmapsto$	$\ell\llbracket P\rrbracket$	
($\mathsf{r\text{-}new}$)	$k\llbracket(\nu e{:}\mathrm{T})\,P\rrbracket$	$\longmapsto$	$(\nu_k e{:}\mathrm{T})\,k\llbracket P\rrbracket$	if $e \neq k$
($\mathsf{r\text{-}split}$)	$k\llbracket P \,\vert\, Q\rrbracket$	$\longmapsto$	$k\llbracket P\rrbracket \,\vert\, k\llbracket Q\rrbracket$	
($\mathsf{r\text{-}comm}$)	$k\llbracket a!\langle v\rangle\,P\rrbracket \mid k\llbracket a?(X)\,Q\rrbracket$	$\longmapsto$	$k\llbracket P\rrbracket \mid k\llbracket Q\{\!\vert{}^{v}\!/\!_{X}\vert\!\}\rrbracket$	
($\mathsf{r\text{-}eq_1}$)	$k\llbracket\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rrbracket$	$\longmapsto$	$k\llbracket P\rrbracket$	
($\mathsf{r\text{-}eq_2}$)	$k\llbracket\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\rrbracket$	$\longmapsto$	$k\llbracket Q\rrbracket$	if $u \neq v$

Structural congruence:

($\mathsf{s\text{-}extr}$)	$M \,\vert\, (\nu_k e{:}\mathrm{T})\,N$	$\equiv$	$(\nu_k e{:}\mathrm{T})\,(M \,\vert\, N)$	if $e \notin \mathrm{fn}(M)$
($\mathsf{s\text{-}grbg_1}$)	$(\nu_k e{:}\mathrm{T})\ \mathbf{0}$	$\equiv$	$\mathbf{0}$	
($\mathsf{s\text{-}grbg_2}$)	$k\llbracket\mathsf{stop}\rrbracket$	$\equiv$	$\mathbf{0}$	
($\mathsf{s\text{-}copy}$)	$k\llbracket *P\rrbracket$	$\equiv$	$k\llbracket P\rrbracket \,\vert\, k\llbracket *P\rrbracket$	

Beside each reduction, we have written the axioms used to infer it, omitting mention of the monoid laws. An example of a process Q that uses the received value (z,x) is '$\mathsf{go}\,z.\,x!\langle 1\rangle$', which after the communication becomes '$\mathsf{go}\,k.\,a!\langle 1\rangle$'.

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 [12, §5]. We use uppercase Roman letters to range over types, whose syntax follows:

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

The syntax provides types for base values, locations, local resources and tuples. Types of the form $\mathrm{K}[\widetilde{\mathrm{A}}]$ are *dependent* tuple types, which allow communication of non-local resources; we discuss these further in the next subsection.

We require that each resource name and variable in a location type appear at most once. Location types are essentially the same as standard record types, and we identify location types up to re-ordering 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 '$\mathsf{loc}\{\,\}$'. Note that in general location types may contain variables. This is convenient for typing, but in the syntax of threads and networks, given in Table 1, we restrict all types to be *closed*, *i.e.* no variables can appear in location types in terms.

The subtyping preorder ($\mathrm{T} <: \mathrm{S}$) is discussed at length in [12]. 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:

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

On tuples, the definition is by homomorphic extension:

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

An important property of the subtyping preorder is that it has a partial meet operator $\sqcap$, which will enable us to accumulate typing information associated with identifiers.

Table 3 Standard Typing

Values (rules for base values not shown):

$$\frac{\Gamma(u) <:\ \mathrm{T}}{\Gamma \vdash_w u{:}\mathrm{T}} \qquad \frac{\Gamma(w) <:\ \mathsf{loc}\{u{:}\mathrm{A}\}}{\Gamma \vdash_w u{:}\mathrm{A}}$$

$$\frac{\Gamma \vdash_w u_i{:}\mathrm{T}_i \ \ (\forall i)}{\Gamma \vdash_w \widetilde{u}{:}\widetilde{\mathrm{T}}} \qquad \frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathrm{K}\\ \Gamma \vdash_u \widetilde{v}{:}\widetilde{\mathrm{B}}\end{array}}{\Gamma \vdash_w (u,\widetilde{v}):\mathrm{K}[\widetilde{\mathrm{B}}]}$$

Threads:

$$\frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathsf{loc}\\ \Gamma \vdash_u P\end{array}}{\Gamma \vdash_w \mathsf{go}\,u.P} \qquad \frac{\begin{array}{l}\Gamma \vdash_w u{:}\mathrm{S}\\ \Gamma \vdash_w v{:}\mathrm{T}\\ \Gamma \sqcap \{_w u{:}\mathrm{T}\} \sqcap \{_w v{:}\mathrm{S}\} \vdash_w P\\ \Gamma \vdash_w Q\end{array}}{\Gamma \vdash_w \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

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

$$\frac{\begin{array}{l}\Gamma \vdash_w P\\ \Gamma \vdash_w Q\end{array}}{\Gamma \vdash_w \mathsf{stop},\ P \,|\, Q,\ {*P}} \qquad \frac{\begin{array}{l}e \notin \mathrm{fn}(\Gamma)\\ \Gamma \sqcap \{_w e{:}\mathrm{T}\} \vdash_w P\end{array}}{\Gamma \vdash_w (\nu e{:}\mathrm{T})\, P}$$

Networks:

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

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) $(t \sqcap s) \sqcap r = t \sqcap (s \sqcap r)$
(d) $t \sqcap s = s \sqcap t$

In the last two conditions $=$ refers to partial equality; if one of the expressions is defined the other must also be defined. □

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

Proof. (Outline) On location types this operator is induced by the following equation:

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

For example:

$$\mathsf{loc}\{a{:}\mathrm{A},\, b{:}\mathrm{B}\} \sqcap \mathsf{loc}\{b{:}\mathrm{B},\, c{:}\mathrm{C}\} = \mathsf{loc}\{a{:}\mathrm{A},\, b{:}\mathrm{B},\, c{:}\mathrm{C}\}$$ □

2.4 Standard Typing

Judgments in the typing system take three forms:

$\Gamma \vdash N$	Network N is well-typed
$\Gamma \vdash_w P$	Thread P is well-typed to run at location w
$\Gamma \vdash_w v{:}\mathrm{T}$	Value v can be assigned type T at location w

Here Γ and Δ range over *type environments*, which map location names to location types and variables to base types or location types.[2] For example, the following is a type environment:

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

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 3. 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 3, 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; *e.g.* 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 $(u,\widetilde{v})$, the typing of $\widetilde{v}$ is deduced with respect to the location identifier u. Thus if $\Gamma \vdash_w (k,a):(\mathrm{K},\mathrm{A})$ then k and a are two independent values whose type consistency is checked independently. On the other hand if $\Gamma \vdash_w (k,a):\mathrm{K}[\mathrm{A}]$ then a is considered to be a resource at the location k, and this judgment depends on a be well-typed at k, $\Gamma \vdash_k a{:}\mathrm{A}$. We emphasize this use of dependent types with some notation:

Notation. In examples, we use $u[\widetilde{v}] \overset{def}{=} (u,\widetilde{v})$ to indicate that the tuple $(u,\widetilde{v})$ has a dependent type, $\mathrm{K}[\widetilde{\mathrm{A}}]$. □

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

The rules for input and restriction are intuitive, although they make use of some notation for updating environments, defined in Appendix A. Suppose, for example, we wish to infer the judgment $\Gamma \vdash_w u?(X{:}\mathrm{T})\,Q$. By α-conversion we may assume that the variables in the pattern X are distinct from those in the environment Γ. So we need to establish two facts:

- relative to Γ, the identifier u is a resource at the location w, *i.e.* $\Gamma \vdash_w u{:}\mathsf{res}\langle\mathrm{T}\rangle$, and
- the continuation Q is well-typed relative to Γ augmented with the typing information in $X{:}\mathrm{T}$.

This extension of the type environment is represented as $\Gamma \sqcap \{_w X{:}\mathrm{T}\}$, the meet of Γ and the environment constructed from X and T relative to w, $\{_w X{:}\mathrm{T}\}$. For example if $X{:}\mathrm{T}$ has the form $x{:}\mathrm{A}$ then the extra information added to Γ is the environment $\{_w x{:}\mathrm{A}\} = \{w{:}\mathsf{loc}\{x{:}\mathrm{A}\}\}$; in typing Q we are therefore allowed to assume that x is a resource of type A local to w. On the other hand if $X{:}\mathrm{T}$ has the form $z[x]{:}\mathrm{K}[\mathrm{A}]$ then the extra information added to Γ is the environment $\{_w z[x]{:}\mathrm{K}[\mathrm{A}]\} = \{z{:}\mathrm{K} \sqcap \mathsf{loc}\{x{:}\mathrm{A}\}\}$. Here when typechecking Q we can assume that x is a resource of type A at location z, which in turn has the type $\mathrm{K} \sqcap \mathsf{loc}\{x{:}\mathrm{A}\}$.

Similarly the network $(\nu_k a{:}\mathrm{A})\,N$ is well-typed with respect to Γ if a does not appear in Γ and N is well-typed with respect to $\Gamma \sqcap \{k{:}\mathsf{loc}\{a{:}\mathrm{A}\}\}$, since $\{_k a{:}\mathrm{A}\}$ works out to be $\{k{:}\mathsf{loc}\{a{:}\mathrm{A}\}\}$. Also proving $\Gamma \vdash_w (\nu_k \ell{:}\mathrm{L})\,N$ involves establishing $\Gamma \sqcap \{\ell{:}\mathrm{L}\} \vdash_w N$ since $\{_k \ell{:}\mathrm{L}\} = \{\ell{:}\mathrm{L}\}$.

[2] For simplicity, the typing system defined here requires that every tuple be fully decomposed upon reception; *i.e.*, 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.

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:

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

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

$$a?(z[x])\ b?(w[y])\ \textsf{if } z = w \textsf{ then go} z.\,\big(x?(u)\ y!\langle u\rangle\big)$$

This thread receives two remote channels from different sources, then forwards messages from one channel to the other. Further examples are given in [12] 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 specialization, weakening and a substitution lemma, as described in [12]. The following result establishes the fact that well-typedness is preserved by reduction. Together with a Type Safety theorem, again described in [12], this ensures that well-typed terms are free of runtime errors throughout their execution.

Theorem 3 (Subject Reduction for the Standard Semantics). *If* $\Gamma \vdash N$ *and* $N \longrightarrow N'$ *then* $\Gamma \vdash N'$. □

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 behavior. 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:

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

This reflects the fact that channels at an untyped location may have any type and consequently behavior at bad locations is unconstrained. With the addition of lbad, the partial meet operator becomes total on location types.[3]

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

Table 4 Partial Typing Relation

All rules from Table 3 but those for restriction (ν)

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

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

Type environments are now more expressive. A typical example is given by:

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

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 three channels: c, which communicates dependent tuples with the second element being an integer channel; b, which communicates dependent tuples with the second element being boolean channels; and d which communicates untyped locations. We will use this environment in most subsequent examples in this section and the next.

The typing relation given in Table 3 may now be applied to this extended language of partial types with the result that untyped locations enjoy many expected properties. For example, since $\mathsf{lbad} <: \mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}$ and $\mathsf{lbad} <: \mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{bool}\rangle\}$, 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 unconstrained by typing considerations; this is the case even if the resource is restricted.

Agents can also use the type information to infer that a remote location is untyped. For example consider the network

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

which is well-typed with respect to Γ, from (*). If an agent receives the same dependent pair (say $n[a]$) on both the channels b and c, then it can determine that location n is untyped. Thus the agent can subsequently output n 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. We would like to have that $\Gamma \vdash m[\![\textsf{go} k.\,a!\langle \mathsf{t}\rangle]\!]$. Here an untyped agent at m attempts to move to k and misuse the integer channel a by sending on it the boolean value t. The standard typing rule for movement, however, does not allow this judgment, since it 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 4. 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. The introduction of the rule $(\mathsf{t\text{-}bad})$ allows untyped locations to have truly arbitrary behavior, including the ability to (attempt to) send malicious agents to good locations. Thus the partial typing relation validates the judgment $\Gamma \vdash m[\![\mathsf{go}\, k.\, a!\langle \mathsf{t}\rangle]\!]$; here the malicious site m attempts to send to the good site k an agent which will misuse the resource a.

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

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

The rules $(\mathsf{t\text{-}new_g})$ and $(\mathsf{n\text{-}new_g})$ 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 12.

3.2 An Example

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:

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

Here the agents at ℓ and k are all quite reasonable; they could be typed using the standard type system of Table 3. 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 $\Gamma \vdash N$, where Γ is as in (*). This well typing, however, is not preserved by reduction.

The agents communicating on c reduce unproblematically, first with a move from k to ℓ, then a communication, then a move from ℓ to k. All of these reductions preserve well-typing under Γ.

The agents communicating on b evolve in the same way, the only difference being that the first move is from m, rather than from k. Using standard reduction (Table 2), we have:

$$\begin{aligned} & \ell[\![b?\langle z[x]\rangle\ \mathsf{go}\,z.\,x!\langle \mathsf{t}\rangle]\!] \mid m[\![\mathsf{go}\,\ell.\,b!\langle k[a]\rangle]\!] && (1)\\ \longrightarrow\; & \ell[\![b?\langle z[x]\rangle\ \mathsf{go}\,z.\,x!\langle \mathsf{t}\rangle]\!] \mid \ell[\![b!\langle k[a]\rangle]\!] && (2)\\ \longrightarrow\; & \ell[\![\mathsf{go}\,k.\,a!\langle \mathsf{t}\rangle]\!] && (3)\\ \longrightarrow\; & k[\![a!\langle \mathsf{t}\rangle]\!] && (4) \end{aligned}$$

Here, however, (2)–(4) are not well-typed under Γ. This fact is obvious when considering (4) where an agent at k attempts to send a boolean on an integer channel. Already in (2), however, typing under Γ fails. In order to infer $\Gamma \vdash \ell[\![b!\langle k[a]\rangle]\!]$ we must establish that for some T, $\Gamma \vdash_{\ell} b{:}\mathsf{res}\langle \mathrm{T}\rangle$ and $\Gamma \vdash_{\ell} k[a]{:}\mathrm{T}$. Given the type of b at ℓ, we would have to take $\mathrm{T} = \mathsf{loc}[\mathsf{res}\langle \mathsf{bool}\rangle]$, but $\Gamma \nvdash_{\ell} k[a]{:}\mathsf{loc}[\mathsf{res}\langle \mathsf{bool}\rangle]$, since a is an integer channel at k.

The semantics presented in the following section will prevent the reduction of (1) to (2) 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, *i.e.* 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 [13]. In anonymous networks, the authority of incoming threads is not known. The semantics of [13] uses a weaker typing system requiring consistency only of *local* resource types. Thus, in that work, (2) is taken to be well-typed, with subject reduction failing only in the move from (3) to (4). The chief advantage of the current work is that it permits the use of *trust*, which appears to be incompatible with terms such as (2).

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 (Table 1) 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. Intuitively each location k should have exactly one filter associated with it. This constraint could be formalized within the typing system, but for simplicity we prefer to treat it separately.

Definition 4. 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. □

We refer to networks with filters as *open networks* and for the rest of the paper, we assume they are always well-formed.

Static Typing The static typing relation extends that of Tables 3 and 4 with the two rules for filters, given in Table 5. The rule $(\mathsf{n\text{-}filter_g})$ 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 $(\mathsf{n\text{-}filter_b})$ 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.

Table 5 Typing and reduction using filters

Static typing: all rules from Table 4

$$(\mathsf{n\text{-}filter_g})\ \frac{\Gamma <: \Delta \quad \Gamma(k) = \Delta(k)}{\Gamma \vdash k\langle\!\langle \Delta \rangle\!\rangle} \qquad (\mathsf{n\text{-}filter_b})\ \frac{\Gamma(k) = \mathsf{lbad}}{\Gamma \vdash k\langle\!\langle \Delta \rangle\!\rangle}$$

Reduction: $(\mathsf{r\text{-}split})$, $(\mathsf{r\text{-}eq_1})$ and $(\mathsf{r\text{-}eq_2})$ from Table 2

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

Dynamic typing: all rules from Table 4, '$\Vdash^{k}_{w}$' replacing '$\vdash_{w}$'

$$(\mathsf{v_f\text{-}self_1})\ \frac{\mathsf{lbad} <: \mathrm{K}}{\Delta \Vdash^{k}_{w} k{:}\mathrm{K}} \qquad (\mathsf{v_f\text{-}self_2})\ \frac{}{\Delta \Vdash^{k}_{k} a{:}\mathrm{A}}$$

$$(\mathsf{t_f\text{-}return})\ \frac{}{\Delta \Vdash^{k}_{w} \mathsf{go}\,k.P}$$

Reduction The reduction relation for open networks is also given in Table 5. The purpose of filters is to check that incoming agents are well-typed. Thus, the main change to the semantics is the replacement of the reduction rule $(\mathsf{r\text{-}move})$ with:

$$\ell\langle\!\langle \Delta \rangle\!\rangle \mid k[\![\mathsf{go}\,\ell.P]\!] \longmapsto \ell\langle\!\langle \Delta \rangle\!\rangle \mid \ell[\![P]\!] \qquad \text{if } k = \ell \text{ or } \Delta \Vdash^{k}_{\ell} P$$

Here $\Delta \Vdash^{k}_{\ell} P$ is a *dynamic typing relation*, discussed below, which intuitively says that P is well-formed to move to location ℓ, if acting under *authority of* k. Agents originating locally are assumed to be well-typed and therefore need not be checked dynamically.

As a network evolves, the filter at a site should be augmented to reflect that sites 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. One possibility is that information is extracted from values which are communicated at the site: 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. The rule $(\mathsf{r_f\text{-}comm})$ formalizes this idea. Alternatives are discussed in the full version.

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 (where the filter at k is omitted):

$$k[\![(\nu a)\,\mathsf{go}\,\ell.\,b!\langle k[a]\rangle]\!] \mid \ell[\![b?(z[x])\,P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle \tag{5}$$

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 — $(\mathsf{r_f\text{-}move})$ 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, *i.e.* at their "home" location. Intuitively this is true because, under this discipline, "bad" agents can only "lie" about resources located at their authority, which must have been a bad location to begin with. Lies about bad locations don't hurt well-typing, since bad locations are untyped.

Formally, the rules for runtime typing extend those of the static type system given in Tables 3 and 4 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 $(\mathsf{v_f\text{-}self_1})$ 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 $(\mathsf{v_f\text{-}self_2})$ allows an incoming agent to refer to resources at its authority. As an example, let $\Delta_\ell = \{\ell{:}\mathsf{loc}\{a{:}\mathsf{res}\langle \mathrm{K[B]}\rangle\}\}$. Although we cannot infer that $\Delta_\ell \vdash_{\ell} a!\langle k[b]\rangle$ using the static typing system, we can deduce $\Delta_\ell \Vdash^{k}_{\ell} a!\langle k[b]\rangle$ using the dynamic typing relation. Thus the following reduction is allowed by the semantics:

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

The rule $(\mathsf{t_f\text{-}return})$ 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 Tables 3 and 4 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.

4.2 Examples

Example 5. First we show how filters are updated via communication with imported agents. Consider the open network (5) discussed above, where the location k wishes to transmit to ℓ the name of a new local resource a of type A. Suppose the filter at ℓ is $\Delta = \{\ell{:}\mathsf{loc}\{b{:}\mathsf{res}\langle \mathrm{K[A]}\rangle\}\}$, so that ℓ has no information about location

k. Then we have the following reductions (we use Γ to represent the filter at k, the contents of which are not important for the example):

$$\begin{array}{l}
k\langle\!\langle\Gamma\rangle\!\rangle \mid k\llbracket(\nu a{:}\mathrm{A})\,\mathsf{go}\,\ell.\,b!\langle k[a]\rangle\rrbracket \mid \ell\llbracket b?(z[x]{:}\mathrm{K}[\mathrm{A}])\,P\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle \\
\longrightarrow \\
(\nu_k a{:}\mathrm{A})\;\; k\langle\!\langle\Gamma'\rangle\!\rangle \mid k\llbracket\mathsf{go}\,\ell.\,b!\langle k[a]\rangle\rrbracket \mid \ell\llbracket b?(z[x]{:}\mathrm{K}[\mathrm{A}])\,P\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle \\
\longrightarrow \\
(\nu_k a{:}\mathrm{A})\;\; k\langle\!\langle\Gamma'\rangle\!\rangle \mid \ell\llbracket b!\langle k[a]\rangle\rrbracket \mid \ell\llbracket b?(z[x]{:}\mathrm{K}[\mathrm{A}])\,P\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle \\
\longrightarrow \\
(\nu_k a{:}\mathrm{A})\;\; k\langle\!\langle\Gamma'\rangle\!\rangle \mid \ell\llbracket\mathsf{stop}\rrbracket \mid \ell\llbracket P\{\!\!\{{}^{k,a}\!/\!_{z,x}\}\!\!\}\rrbracket \mid \ell\langle\!\langle\Delta'\rangle\!\rangle
\end{array}$$

where $\Gamma' = \Gamma \sqcap \{k{:}\mathsf{loc}\{a{:}\mathrm{A}\}\}$ and $\Delta' = \Delta \sqcap \{k{:}\mathsf{loc}\{a{:}\mathrm{A}\}\}$.

The first move, using the rule ($\mathsf{r_f}$-newr), extrudes the local resource a at k, updating the filter at k accordingly. Using the structural congruence, the scope of a can be extended to include the entire network. Then ($\mathsf{r_f}$-move) may be employed since $\Delta \Vdash^{k}_{\ell} b!\langle k[a]\rangle$. Here the dynamic typing by the filter Δ at ℓ of the incoming thread $b!\langle k[a]\rangle$ succeeds essentially because of the rule ($\mathsf{v_f}$-self$_2$); the thread only communicates the names of resources at k, its authority. Finally a local communication at ℓ is performed, using rule ($\mathsf{r_f}$-comm). Not only is the value communicated to P, but the filter at ℓ is also updated; after the communication, the filter for ℓ contains information about the type of resource a at k. □

Example 6. Let us now revisit open network (1) 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 (n-$\mathsf{filter_g}$). Thus, let us consider the open network

$$\Gamma \vdash m\llbracket\mathsf{go}\,\ell.\,b!\langle k[a]\rangle\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle$$

where Γ is given by (*) in Section 3.1. Note that the agent at m attempts to misinform and 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 $\Delta \Vdash^{m}_{\ell} b!\langle k[a]\rangle$, that is if we can dynamically typecheck $b!\langle k[a]\rangle$ using the filter Δ under the authority m. But this is impossible, given the constraint that $\Gamma \vdash \ell\langle\!\langle\Delta\rangle\!\rangle$. To see this, first note that ℓ has full self-knowledge, *i.e.* $\Delta(\ell) = \Gamma(\ell)$, and therefore $\Delta(\ell)$ must have the entry $b{:}\mathsf{res}\langle\mathsf{loc}[\mathsf{res}\langle\mathsf{bool}\rangle]\rangle$; 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 3 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 5 be used to justify the claim that $\Delta \Vdash^{m}_{k} a{:}\mathsf{res}\langle\mathsf{bool}\rangle$. It follows that the inference $\Delta \Vdash^{m}_{\ell} b!\langle k[a]\rangle$ is impossible. □

Example 7. 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. As an example suppose the thread at location m is changed to $m\llbracket\mathsf{go}\,\ell.\,b!\langle m[a]\rangle\rrbracket$, *i.e.* m wishes to inform ℓ of a resource local to m. Then we have the reduction:

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

This follows since $\Delta \Vdash^{m}_{\ell} m[a]{:}\mathsf{loc}[\mathsf{res}\langle\mathsf{bool}\rangle]$ can be inferred using ($\mathsf{v_f}$-self$_1$) and ($\mathsf{v_f}$-self$_2$), regardless of the type assigned to m in Δ. □

Example 8. An untyped site will also succeed in sending an agent if the reception site already knows the information being received. For example suppose ℓ's filter is extended so that ℓ knows the type of resource a at k, that is $\Delta(k) = \mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}$. Then we have the reduction

$$m\llbracket\mathsf{go}\,\ell.\,c!\langle k[a]\rangle\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle \longrightarrow \ell\llbracket c!\langle k[a]\rangle\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle$$

because of the inference $\Delta \Vdash^{m}_{\ell} c!\langle k[a]\rangle$. Of course the authority of m plays no role in this judgment. □

Example 9. The information in filters determines 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 open network, again typed using the environment Γ given in (*), where Δ is the restriction of Γ onto ℓ, *i.e.* $\Delta = \{\ell{:}\Gamma(\ell)\}$:

$$m\llbracket\mathsf{go}\,\ell.\,c!\langle k[a]\rangle\rrbracket \mid k\llbracket\mathsf{go}\,\ell.\,c!\langle k[a]\rangle\rrbracket \mid \ell\llbracket *c?(z[x])\,P\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle$$

Here the migration from m to ℓ is not immediately possible, since $\Delta \nVdash^{m}_{\ell} c!\langle k[a]\rangle$. However the migration from k is allowed since $\Delta \Vdash^{k}_{\ell} c!\langle k[a]\rangle$, and the network reduces, after communication on c, to:

$$m\llbracket\mathsf{go}\,\ell.\,c!\langle k[a]\rangle\rrbracket \mid \ell\llbracket P'\rrbracket \mid \ell\llbracket *c?(z[x])\,P\rrbracket \mid \ell\langle\!\langle\Delta'\rangle\!\rangle$$

where $P' = P\{\!\!\{{}^{k,a}\!/\!_{z,x}\}\!\!\}$ and $\Delta' = \Delta \sqcap \{k{:}\mathsf{loc}\{a{:}\mathsf{res}\langle\mathsf{int}\rangle\}\}$. The migration from m to ℓ can now take place, allowing the network to reduce, after a further communication, to:

$$\ell\llbracket P'\rrbracket \mid \ell\llbracket P'\rrbracket \mid \ell\llbracket *c?(z[x])\,P\rrbracket \mid \ell\langle\!\langle\Delta'\rangle\!\rangle$$

since $\Delta' \Vdash^{m}_{\ell} c!\langle k[a]\rangle$. In the absence of other agents, the migrations can only be executed in one order (k first). □

Example 10. 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 assumed to be bad. As an example let Γ and the filter $\Delta = \{\ell{:}\Gamma(\ell)\}$ be as before, and consider the open network:

$$m\llbracket\mathsf{go}\,\ell.\,b!\langle m[d]\rangle\,c!\langle m[d]\rangle\rrbracket \mid \ell\llbracket b?(z[x]{:}\mathrm{T})\,c?(w[y])\,P\rrbracket \mid \ell\langle\!\langle\Delta\rangle\!\rangle$$

where T is the same type as b at ℓ, $\mathsf{res}\langle\mathsf{loc}[\mathsf{res}\langle\mathsf{bool}\rangle]\rangle$. After the migration from m to ℓ and one communication this reduces to

$$\ell\llbracket c!\langle m[d]\rangle\rrbracket \mid \ell\llbracket c?(w[y])\,P'\rrbracket \mid \ell\langle\!\langle\Delta'\rangle\!\rangle$$

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

$$\ell\llbracket P''\rrbracket \mid \ell\langle\!\langle\Delta''\rangle\!\rangle$$

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

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 5.

Theorem 11 (Subject Reduction for Open Networks). *For the inference systems of Table 5: If $\Gamma \vdash N$ and $N \longrightarrow N'$ then $\Gamma \vdash N'$.* □

Table 6 Runtime Error

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

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

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 [12, 21]. However in this case the presence of filters makes it straightforward.

In Table 6 we define, for each location ℓ a unary predicate $\xrightarrow{err\,\ell}$ over networks. The judgment $N \xrightarrow{err\,\ell}$ should be read: "in the network N there is a runtime error at location ℓ". There are two kinds of errors which can occur. The first occurs when an agent attempts to use a resource that has not been allocated at the agent's current location, as formalized in the first two clauses of the definition in Table 6. 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 6, 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 6). This fact explains the need for the side condition $\mathrm{T} \neq \mathsf{lbad}$ on the rules ($\mathsf{t\text{-}new}_\mathsf{g}$) and ($\mathsf{n\text{-}new}_\mathsf{g}$) in Table 4.

Theorem 12 (Type Safety for Open Networks). *For the inference systems of Tables 5 and 6: If $\Gamma \vdash N$ and $\Gamma(\ell) \neq \mathsf{lbad}$ then $N \not\xrightarrow{err\,\ell}$.* □

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 optimization which allows for freer and more efficient movement across the network. 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 with trust* is obtained by replacing the clause for location types with:

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

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 location names and others for handling untrusted location names. In a similar vein we may have untrusted locations containing resources that communicate trusted location names. 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 behavior; 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:

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

The formal definition is given in the full version of the paper.

Proposition 13. *The set of types with trust, under the subtyping preorder, has a partial meet operator.* □

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 now three possibilities with respect to a remote location k mentioned in $\ell\langle\!\langle\Delta\rangle\!\rangle$:

- $\Delta(k) <: \mathsf{lbad}$, which means that ℓ has accumulated sufficient contradictory information about k to conclude that k is untyped.
- $\Delta(k) <: \mathsf{ltrust}$, 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 ($\mathsf{n\text{-}filter}$) in Table 5 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 $\mathsf{lbad} \not<: \mathsf{ltrust}$.
- $\Delta(k) <: \mathsf{loc}$, 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, *i.e.* $\ell\langle\!\langle\Delta\rangle\!\rangle$ 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 Example 10 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 5, replacing ($\mathsf{r_f\text{-}move}$) with:

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

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

Example 14. Let $\Delta = \{\ell{:}\mathsf{loc}\{d{:}\mathsf{res}\langle\mathsf{ltrust}\rangle\}, k{:}\mathsf{ltrust}\}$ and consider the open network:

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

Here the locations m and n are unknown to ℓ, *i.e.* $\Delta(m)$ 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 $\Delta \Vdash^{m}_{\ell} d!\langle n\rangle$ 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

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

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 15. Consider the network

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

where there is a web of trust among ℓ_i; that is $\Delta_i(\ell_j) <: \mathsf{ltrust}$ 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 judgment can be verified:

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

Note that this checks not only the potential behavior 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

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

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 4, we gave an example which shows that such typechecking is necessary for agents which wish to reenter a web of trust. □

Example 16. 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 main results of the previous section extend to the new setting.

Theorem 17. *For the inference systems of Tables 5 and 6, augmented with trusted location types and using* $(\mathsf{r_t\text{-}move})$ *instead of* $(\mathsf{r_f\text{-}move})$*:*

- *If* $\Gamma \vdash N$ *and* $N \longrightarrow N'$ *then* $\Gamma \vdash N'$.
- *If* $\Gamma \vdash N$ *and* $\Gamma(\ell) \neq \mathsf{lbad}$ *then* $N \overset{err\,\ell}{\not\longrightarrow}$. □

6 Conclusions

We introduced the notion of *partial typing*, which captures the intuition that "bad" sites in a network may harbor 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. 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 is very different from that in [23] but is only a minor variant on that in [12]; 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 $(\mathsf{r\text{-}new})$ 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, 4], 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. To add partial typing to such a language, one would need to typecheck *message* movement dynamically, rather than thread movement, violating the third principle given in the introduction.

The fact that resource names are allowed to occur at multiple locations is crucial to the success of our strategy for dynamic typechecking. It would be difficult to formulate our approach under the assumption that each name has a unique location (as, for example, in [4]). For example, suppose that the resource a was "uniquely located" at k. Then the agent $m[\![\mathsf{go}\,\ell.\,b!\langle m[a]\rangle]\!]$ at the bad site m could "hijack" a using $(\mathsf{t\text{-}self_2})$, 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 m.

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, *i.e.* resources may be *referenced* at remote sites, but may only be *used* locally.

Cardelli and Gordon's ambient calculus [5], on the other hand, 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 [16], and this sorting system has recently been extended to the ambient calculus [7]. 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 later, our resources. This discipline would open the possibility of typing code during in and out operations, rather than open.

Several studies have addressed the issue of static typing for languages with remote resources; some recent papers are [20, 6, 24]. Perhaps the work closest to ours is that of Knabe [14], who has implemented an extension of Facile which supports mobile agents. The main extensions are remote signatures and proxy structures, which are somewhat related to our location types. None of these works address open systems, however. On the other hand, Necula's proof carrying code [19] and related techniques [27, 15, 18] address the problem of dynamic typechecking in open systems, but do not consider the subject of remote resources.

Another area of related work has to do with static methods for analyzing the security of information flow [9, 1, 8, 26, 11]. Although this area of research share our general aims there is very little technical overlap with our approach to resource protection in open systems.

Acknowledgments We wish to thank Alan Jeffrey for many interesting conversations and the referees for their close reading of the text.

A Environment Extension

Both subtyping and the partial meet operator extend pointwise 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:

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

New environments are created from values using the notation $\{_w u{:}\mathrm{T}\}$, where $w \in \mathit{Loc} \cup \mathit{Var}$. The definition is given by induction on u and T:

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

For example:

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

References

[1] M. Abadi. Secrecy by typing in security protocols. In *Proceedings of TACS97*, volume 1218 of *Lecture Notes in Computer Science*, pages 611–637. Springer-Verlag, 1997.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, To appear. Available as SRC Research Report 149 (1998).

[3] R. Amadio and S. Prasad. Localities and failures. In *Proc. 14th Foundations of Software Technology and Theoretical Computer Science*, volume 880 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[4] Roberto Amadio. An asynchronous model of locality, failure, and process mobility. In *COORDINATION '97*, volume 1282 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[5] L. Cardelli and A. D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Proc. FOSSACS'98, International Conference on Foundations of Software Science and Computation Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, 1998.

[6] Luca Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, January 1995. A preliminary version appeared in Proceedings of the 22nd ACM Symposium on Principles of Programming.

[7] Luca Cardelli and Andrew Gordon. Types for mobile ambients. Draft, 1998. Available from http://www.luca.demon.co.uk/.

[8] Mads Dam. Proving trust in systems of second-order processes. In *Hawaii International Conference on Systems Science*. IEEE Computer Society Press, 1998.

[9] D. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20:504–513, 1977.

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

[11] Nevin Heintz and Jon G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[12] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. Computer Science Technical Report 2/98, University of Sussex, 1998. Extended abstract in HLCL '98. Available from http://www.elsevier.nl/locate/entcs/volume16.3.html.

[13] Matthew Hennessy and James Riely. Type-safe execution of mobile agents in anonymous networks. Computer Science Technical Report 3/98, University of Sussex, 1998. Available from http://www.cogs.susx.ac.uk/.

[14] Frederick Coleville Knabe. *Language Support for Mobile Agents*. PhD thesis, Carnegie-Mellon University, 1995.

[15] Dexter Kozen. Efficient code certification. Technical Report 98-1661, Cornell University, Department of Computer Science, 1988. Available from `http://www.cs.cornell.edu/kozen/secure`.

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

[17] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100:1–77, September 1992.

[18] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From System F to typed assembly language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, pages 85–97, San Diego, January 1998. ACM Press.

[19] George Necula. Proof-carrying code. In *Conference Record of the ACM Symposium on Principles of Programming Languages*. ACM Press, January 1996.

[20] Atsuhi Ohori and Kazuhiko Kato. Semantics for communication primitives in a polymorphic language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, Charleston, January 1993. ACM Press.

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

[22] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. Computer Science Technical Report 4/98, University of Sussex, 1998. Available from `http://www.cogs.susx.ac.uk/`.

[23] James Riely and Matthew Hennessy. A typed language for distributed mobile processes. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[24] Tatsurou Sekiguchi and Akinori Yonezawa. A calculus with code mobility. In *FMOODS '97*, Canterbury, July 1997. Chapman and Hall.

[25] Peter Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proceedings of ICALP '98: International Colloquium on Automata, Languages and Programming (Aarhus)*, number 1443 in LNCS, pages 695–706. Springer-Verlag, July 1998.

[26] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[27] Frank Yellin. Low-level security in Java. In *WWW4 Conference*, 1995. Available from `http://www.javasoft.com/sfaq/verifier.html`.