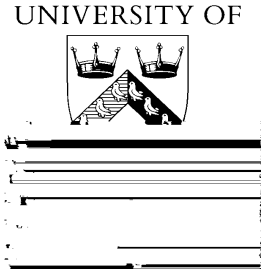
UNIVERSITY OF SUSSEX

COMPUTER SCIENCE



Trust and Partial Typing in Open Systems of Mobile Agents

James Riely and Matthew Hennessy

Trust and Partial Typing in Open Systems of Mobile Agents

JAMES RIELY AND 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 [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*, *e.g.* integer resources are always accessed with integers and string resources are always accessed with strings, and
- resource access is always *authorized*, *i.e.* 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, *e.g.* to "type-check the web". In this paper, we present type systems and semantics for open systems

Trust and Partial Typing in Open Systems of Mobile Agents

corrupted; an example of this phenomenon is described in Section 3.2. Technically

Table 1 Syntax of names e, values u, patterns X, threads P, and networks M.

e	$::=$ k	Location	u,v,w	$::=$ bv	Base Value
	$\mid$ a	Resource		$\mid$ e	Name
X,Y	$::=$ x	Variable		$\mid$ x	Variable
	$\mid$ $(X_1,..,X_n)$	Tuple		$\mid$ $(u_1,..,u_n)$	Tuple
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			

contain the main contributions of the paper. 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*. 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 formalization of the language differs slightly from that of [13], as discussed in the conclusion.

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*.

Names, *e*, include location names and resource names. *Values*, *u–w*, include base values, names, variables and tuples of values. We occasionally use the metavariables *u–w* to range over restricted classes of values, such as $Var \cup Loc$ or $Var \cup Res$;

such cases should be clear from context. *Patterns*, *X*–*Y*, include variables and tu-

Table 2 Standard Reduction

On tuples, the definition is by homomorphic extension:

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

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

DEFINITION 2.1. A partial binary operator $\sqcap$ on a preorder $(S, \preceq)$ is a partial meet operator if it satisfies the following for every $r, s, t \in S$:

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 $\mathtt{goto}\,u.P$ to be well-typed at some location w, P must be well-typed at location u.

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 N is well-typed with respect to Γ extended at k by the type information in declaration $e{:}\mathrm{T}$, *i.e.* $\Gamma \sqcap \{_k e{:}\mathrm{T}\} \vdash N$.

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:

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

This thread receives two remote channels from different sources, then forwards messages from one channel to the other. Further examples are given in [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 specialization, 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

Table 4 Partial Typing Relation

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

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

$$(\mathsf{net\text{-}new_b})\frac{\begin{array}{l}\Gamma(k) = \mathsf{lbad}\\ \ell \notin \mathrm{fn}(\Gamma)\\ \Gamma \sqcap \{\ell{:}\mathsf{lbad}\} \vdash Pw\end{array}}{}$$

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. Most important, the introduction of the rule $(\mathsf{thread\text{-}bad})$ 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 $\Gamma \vdash m[\![\mathsf{goto}\, k.\, a!\langle \mathrm{t} \rangle]\!]$, 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 4.10.

3.2 An Example

Consider the following (partial) type environment:

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

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 dependent tuples with the second element being an integer channel; and b, which communicates dependent tuples with the second element being 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:

$$\begin{array}{rl} P = & \ell[\![c?\langle w[y\ \rangle\ \ \mathsf{goto}\, w.\, y!\langle 0 \rangle]\!] \\ \mid & \ell[\![b?\langle z[x\ \rangle\ \ \ \mathsf{goto}\, z.\, x!\langle \mathrm{t} \rangle]\!] \\ \mid & k[\![\mathsf{goto}\, \ell.\, c!\langle k[a\ \rangle]\!] \\ \mid & m[\![\mathsf{goto}\, \ell.\, b!\langle k[a\ \rangle]\!] \end{array}$$

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 bad), for no Δ do we have $\Delta \vdash P$. 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 P. On the other hand, using the partial typing system, we have $\Gamma \vdash P$. This well typing, however, is not preserved by reduction.

First consider the agents communicating on c. Using standard reduction, as defined in Table 2, these agents reduce as follows.

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

It is worth contrasting this approach with the "purely local" approach adopted for "anonymous networks" in [14] (and outlined in Appendix B). In anonymous networks, the authority of incoming threads is not known. The semantics of [14] 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

Table 5 Typing and reduction using filters

Static typing: all rules from Table 4

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

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

$$\begin{array}{rl}
(\mathsf{r_f\text{-}move}) & k[\![\mathsf{goto}\,\ell.P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle \quad \text{if } k = \ell \text{ or } \Delta \Vdash^k_\ell P \\
& \longmapsto \quad \ell[\![P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle \\[2ex]
(\mathsf{r_f\text{-}newr}) & k[\![(\nu a{:}\mathrm{A})\,P]\!] \mid k\langle\!\langle \Delta \rangle\!\rangle \qquad \text{if } a \notin \mathrm{fn}(\Delta) \\
& \longmapsto (\nu_k a{:}\mathrm{A})\,\big(k[\![P]\!] \mid k\langle\!\langle \Delta \sqcap \{{}_k a{:}\mathrm{A}\} \rangle\!\rangle\big) \\[2ex]
(\mathsf{r_f\text{-}newl}) & k[\![(\nu \ell{:}\mathrm{L})\,P]\!] \mid k\langle\!\langle \Delta \rangle\!\rangle \qquad \text{if } \ell \notin \mathrm{fn}(\Delta) \cup \{k\} \\
& \longmapsto (\nu_k \ell{:}\mathrm{L})\,\big(k[\![P]\!] \mid k\langle\!\langle \Delta \sqcap \{\ell{:}\mathrm{L}\} \rangle\!\rangle \mid \ell\langle\!\langle \{\ell{:}\mathrm{L}\} \rangle\!\rangle\big) \\[2ex]
(\mathsf{r_f\text{-}comm}) & k[\![a!\langle v \rangle\,P]\!] \mid k[\![a?(X{:}\mathrm{T})\,Q]\!] \mid k\langle\!\langle \Delta \rangle\!\rangle \\
& \longmapsto \quad k[\![P]\!] \mid k[\![Q\{{}^{v}/_{X}\}]\!] \mid k\langle\!\langle \Delta \sqcap \{{}_k v{:}\mathrm{T}\} \rangle\!\rangle
\end{array}$$

Dynamic typing: all rules from Table 4, with ‘

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. 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:

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

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 — $(\mathsf{r_f\text{-}move})$ 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 to "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 $(\mathsf{val_f\text{-}self_1})$ 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 $\mathsf{bad} <: \mathrm{K}$ is vacuously satisfied; we include it here only for reference in the next section.) The rule $(\mathsf{val_f\text{-}self_2})$ allows an incoming agent to refer to resources at its authority. As an example, let $\Delta_\ell = \{\ell{:}\mathsf{loc}\{a{:}\mathsf{res}\langle \mathrm{K}[\mathrm{B}\,\rangle\}\}$. Although we cannot infer that $\Delta_\ell \vdash_{\ell} a!\langle$

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 Tables 3 and 4 with respect to an omniscient authority (Γ

$\Delta(\ell) = \Gamma(\ell)$, 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 4.4. 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:

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

This follows since $\Delta \Vdash^m_\ell m[a : \mathsf{loc}[\mathsf{res}\langle\mathsf{bool}\rangle$ can be inferred using $(\mathsf{val_f}\text{-}\mathsf{self}_1)$ and $(\mathsf{val_f}\text{-}\mathsf{self}_2)$

kheo[is inceea00187(s)-1(o)4(ds-343.007(t)-1(i)-4(29987(t)-2(he)4.00187(t)-241.987(r)-2(r

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 4.7. 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:

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

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

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

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

$$\ell[\![P$$

Table 6 Runtime Error

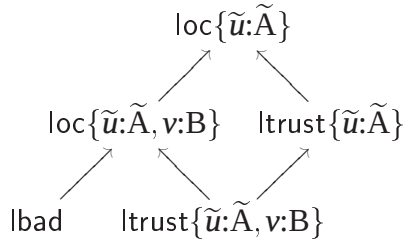
$$
\begin{array}{rcll}
\ell[\![a?(X{:}\mathrm{T})\,P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle & \xrightarrow{err\,\ell} & \text{if } \Delta(\ell) \not<: \mathsf{loc}\{a{:}\mathsf{res}\langle \mathrm{T}\rangle\} & \\
\ell[\![a!\langle v\rangle\,P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle & \xrightarrow{err\,\ell} & \text{if } \Delta(\ell) \not<: \mathsf{loc}\{a{:}\mathsf{res}\langle \mathrm{T}\rangle\}, \text{ all T} & \\
\ell[\![a!\langle v\rangle\,P]\!] \mid \ell\langle\!\langle \Delta \rangle\!\rangle & \xrightarrow{err\,\ell} & \text{if } \Delta(\ell) <: \mathsf{loc}\{a &
\end{array}
$$

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 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:



The formal definition is given in Appendix A.

PROPOSITION 5.1. *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 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 $(\mathsf{net\text{-}filter})$ 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 4.7 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:

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

Note that the presence of ltrust changes the importance of the condition $\mathsf{lbad} <: \mathrm{K}$ in the dynamic typing rule $(\mathsf{val_f\text{-}self_1})$. 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 $\Delta \Vdash^k_\ell k{:}\mathsf{ltrust}$. This is important, as it prevents bad sites from becoming trusted.

EXAMPLE 5.2. 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, *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 Δ

EXAMPLE 5.3. Consider the network

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

rule (r-comm), from the standard semantics. Another possibility would be to add analysis to the filter operation. Then the move rule would become:

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

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", *i.e.* 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 6:

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

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:

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

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

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

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

In practice, 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 $k[a$ is of the appropriate type.

6.4 Anonymous Networks

In [14] we presented a semantics for open system in which the authority of incoming agents is not known. We call such systems *anonymous networks*. In Appendix B we recast the semantics of [14] using filters and bad. An attractive property of the semantics is that filter updating is purely local, *i.e.* no non-local data need be stored in filters. 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.

6.5 Plugins

One quickly discovers a limitation of Dπ when trying to model *mirroring* of names across a network. The idea is to create a new resource, say a class name, at one

static, the second dynamic:

$$\frac{\Gamma \vdash_{\overline{u}} v{:}\mathrm{A} \qquad \Gamma \sqcap \{w{:}\mathsf{loc}\{u.v{:}\mathrm{A}\}\} \vdash_{w} P}{\Gamma \vdash_{w} (\mathsf{load}\, u.v{:}\,\mathsf{class}\,\mathrm{A})P} \qquad\qquad \frac{}{\Delta \Vdash^{k}_{w} k.a{:}\mathsf{class}\,\mathrm{A}}$$

We believe that using indexed names for mirrorable values will be crucial to establish Subject Reduction for such a language under partial typing. Note that such a naming strategy has been adopted by the Java community, although perhaps for different reasons, where class names are of the form com.ibm.aglet.

7 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, 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 $(\mathsf{r\text{-}new})$ 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 [4, 5], 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 [5] appears to be incompatible with partial typing, as outlined at the end of Section 4.1.

The join calculus of Fournet, Gonthier, Levy, Marganget and Remy [11] shares many of these properties. Whereas Amadio's language adds thread movement to message movement, however, the join calculus adds location movement. Unfortu-

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 5.1.

The formal definition of subtyping with lbad and ltrust is:

$$\mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}}, \widetilde{v}{:}\widetilde{\mathrm{T}}\} <: \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \qquad \mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}}, \widetilde{v}{:}\widetilde{\mathrm{T}}\} <: \mathsf{ltrust}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\}$$
$$\mathsf{lbad} <: \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\} \qquad \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}, \widetilde{v}{:}\widetilde{\mathrm{T}}\} <: \mathsf{loc}\{\widetilde{u}{:}\widetilde{\mathrm{S}}\}$$
$$\mathsf{lbad} <: \mathsf{lbad}$$

PROPOSITION (5.1). ***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):

$$\mathsf{lbad} \sqcap \mathsf{lbad} = \mathsf{lbad}$$
$$\mathsf{lbad} \sqcap \mathsf{loc}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} = \mathsf{lbad}$$
$$\mathsf{lbad} \sqcap \mathsf{ltrust}\{\widetilde{v}{:}\widetilde{\mathrm{T}}\} =$$

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 $(\mathsf{r_f\text{-}comm})$ and $(\mathsf{r_f\text{-}move})$, which we discuss below.

For the most part, the proof for $(\mathsf{r_f\text{-}comm})$ 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 $\Gamma <: \Delta$. We must show that $\Delta' = \Delta \sqcap \{_k v{:}\mathrm{T}\}$ is defined and that $\Gamma <: \Delta'$, but this follows immediately from Lemma 2.3c and Lemma 2.3a.

Now let us turn to $(\mathsf{r_f\text{-}move})$. 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 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:

Note that the definition of static typing here is much weaker than that presented in the body of the paper. For example the network (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 [14]).

This formulation has certain advantages over that of [14], such as the stronger language of partial types. Moreover it allows self moves to go untyped; *i.e.* reductions of the form $\ell[\![\mathtt{goto}\,\ell.P]\!] \longmapsto \ell[\![P]\!]$ are always allowed.

References

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

[2] Martín Abadi. Secrecy by typing in security protocols. Draft, 1997. Available from `ttp://www.researc .digital.com/ RC/personal/Martin_ Abadi/ ome. tml`.

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

[4] 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.

[5] 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.

[6] L. Cardelli and A. D. Gordon. Mobile ambients, 1997. Draft, Available from `ttp://www.cl.cam.ac.uk/users/adg/`.

[7] Luca Cardelli. A language with distributed scope.