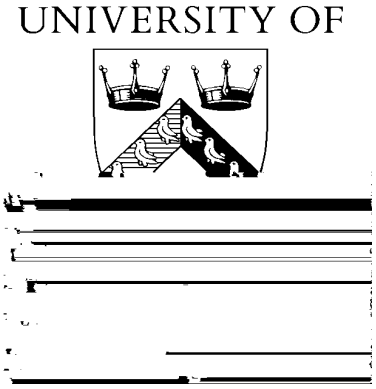
UNIVERSITY OF SUSSEX

COMPUTER SCIENCE



Type-Safe Execution of Mobile Agents in Anonymous Networks

Matthew Hennessy and James Riely

Report 3/98 May 1998

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

ISSN 1350–3170

Type-Safe Execution of Mobile Agents in Anonymous Networks

MATTHEW HENNESSY AND JAMES RIELY

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

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

1 Introduction

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

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

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

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

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

TABLE 1 Syntax

$$\begin{array}{rrcl} \mathit{Id}\colon & u, v, w & ::= & e \mid x \\ \mathit{Val}\colon & U, V & ::= & i \mid \mathsf{bv} \mid u \mid w[u \end{array}$$

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

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

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

- In the concrete syntax, "move" has greater binding power than composition. Thus $\ell :: p \mid q$ should be read $(\ell :: p$

TABLE 2 Standard Reduction

Structural equivalence:

$(\mathsf{s\text{-}nil})$ $\quad$ $\ell \triangleright\!\triangleright \ell \triangleright\!\triangleright\!\triangleright\!\triangleright\!\triangleright\!\triangleright\!\triangleright\!\triangleright\!\triangleright)\mathsf{s\_}$

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

2.3 Types and Subtyping

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

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

Types are divided into the following syntactic groups:

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

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

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

Location types are essentially the same as standard record types, and we identify location types up to reordering of their "fields". Thus $\mathsf{loc}\{a{:}\mathrm{A}, b{:}\mathrm{B}\}[\mathrm{C}] = \mathsf{loc}\{b{:}\mathrm{B}, a{:}\mathrm{A}\}[\mathrm{C}]$. But reordering is not allowed on "abstract" fields. Thu5.9984(t).2u(3

object types:

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

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

2.4 Type Environments

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

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

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

The subtyping relation extends directly to open location types:

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

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

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

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

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

In the typing system of the next section we need some notation for extending (open) type environments conveniently:

TABLE 4 Typing for Values and Threads

Values:

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

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

Threads:

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

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

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

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

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

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

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

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

3.1 Runtime Typing

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

references to local resources. Thus judgments are of the form

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

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

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

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

are always allowed, regardless of the information in Δ.

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

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

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

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

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

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

is transmitted locally on a its validity is not checked.

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

3.2 An Example

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

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

Let P be the following system:

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

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

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

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

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

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

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

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

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

However the next potential move, the migration of the thread $a!\langle \mathrm{t} \rangle$ from ℓ to k,

TABLE 5 Static typing

Values and Threads: As in Table 4

Systems:

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

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

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

$$\Gamma \vdash P$$

where Γ is a (open) type environment and P a system. The type environment only records the types of good locations; thus $k \in \mathrm{dom}(\Gamma)$ is to be read "k is good" and m

TABLE 6 Runtime Error

$$\begin{array}{rll}
(\text{e-comm}) & \ell[\![a!\langle V\rangle\, p]\!] \mid \ell[\![a?(X{:}\mathrm{T})\, q]\!] \xrightarrow{err\,\ell} & \text{if } V \not\asymp \mathrm{T} \\
(\text{e-eq}) & \ell[\![\mathsf{if}\ U = V\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] \xrightarrow{err\,\ell} & \text{if } U \not\asymp V \\
(\text{e-new}) & P \not\stackrel{err\,k}{\Longrightarrow} &
\end{array}$$

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

THEOREM

extension of *partial* typing to these richer types.

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

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

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

A Proofs

A.1 Properties of the static type system

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

PROPOSITION A.1 (TYPE SPECIALIZATION).
If $\mathbb{L} \vdash V{:}\mathrm{T}$ *and* $\mathrm{T} \leq \mathrm{S}$ *then* $\mathbb{L} \vdash V{:}\mathrm{S}$.

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

PROPOSITION A.2 (WEAKENING).
- *If* $\mathbb{L} \vdash V{:}\mathrm{T}$ *and* $\mathbb{K} \leq \mathbb{L}$ *then* $\mathbb{K} \vdash V{:}\mathrm{T}$
- *If* $\mathbb{L} \vdash p$ *and* $\mathbb{K} \leq \mathbb{L}$ *then* $\mathbb{K} \vdash p$
- *If* $\Gamma, w{:}\mathbb{L} \vdash P$ *and* $\mathbb{K} \leq \mathbb{L}$ *then* $\Gamma, w{:}\mathbb{K} \vdash P$

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

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

We present four cases for the proof on systems.

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

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

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

(r-new). We consider the case:

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

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

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

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

A.3 Type Safety

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

LEMMA A.9.

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

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

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

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

(e-comm). In this case we have $\ell[\![a!\langle V\rangle\, p]\!] \mid \ell[\![a?(X{:}\mathrm{T})\, q]\!] \xrightarrow{err\,\ell}$

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

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

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

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