Information Flow vs. Resource Access in the Asynchronous Pi-Calculus

MATTHEW HENNESSY and JAMES RIELY

ABSTRACT. We propose an extension of the asynchronous π-calculus in which a variety of security properties may be captured using types. These are an extension of the input/output types for the π-calculus in which i/o capabilities are assigned specific security levels. The main innovation is a uniform typing system which, by varying slightly the allowed set of types, captures different notions of security.

We first define a typing system which ensures that processes running at security level σ cannot access resources with a security level higher than σ. The notion of *access control* guaranteed by this system is formalized in terms of a Type Safety Theorem.

We then show that, by restricting the allowed types, our system prohibits implicit information flow from high-level to low-level processes. We prove that low-level behaviour can not be influenced by changes to high-level behaviour. This is formalized as a *non-interference* theorem with respect to may testing.

KEYWORDS: pi-calculus, security, noninterference, may testing, I/O subtyping.

1 Introduction

The problem of protecting information and resources in systems with multiple sensitivity or security levels, [2], has been studied extensively. Flow analysis techniques have been used in [3, 4], axiomatic logic in [14] while in [29, 16] type systems have been developed for a number of prototypical programming languages. In this paper, we explore the extent to which typing systems for ensuring various forms of security can also be developed for the asynchronous π-calculus [5, 18]. Specifically we show that the standard typing system for the picalculus can be extended in a simple manner so as to address these issues. By varying the types used two quite separate security issues can be addressed: resource access control and information control. The former is described in terms of runtime errors, the latter in terms of non-interference [29, 12].

The (asynchronous) π-calculus is a very expressive language for describing distributed systems, [5, 24, 13], in which processes intercommunicate using channels. Thus $n?(x)\,P$ is a process which receives some value on the channel named n, binds it to the variable x and executes the code P. Corresponding to this input command is the asynchronous output command $n!\langle v\rangle$ which outputs the value v on n. The set of values which may be transmitted on channels includes channel names themselves; this, together with the ability to dynamically create new channel names, gives the language its descriptive power.

Within the setting of the π-calculus we wish to investigate the use of types to enforce security policies. To facilitate the discussion we extend the syntax with a new construct to represent a process running at a given security clearance, $\sigma[\![P]\!]$. Here σ is some security level taken from a complete lattice of security levels SL and P is the code of the process. Further, we associate with each channel, the *resources* in our language, a set of input/output capabilities [23, 25], each decorated with a specific security level. Intuitively, if channel n has a read capability at level σ, then only processes running at security level σ or higher may be read from n. This leads to the notion of a *security policy* Σ, which associates a set of capabilities with each channel in the system. The question then is to design a typing system which ensures that processes do not violate the given security policy.

Of course this depends on when we consider such a violation to take place. For example if Σ assigns the channel or resource n the highest security level top then it is reasonable to say that a violation will eventually occur in

$$c!\langle n\rangle \quad | \quad \mathsf{bot}[\![c?(x)\;x?(y)\,P]\!]$$

as after the communication on c, a low level process, $\mathsf{bot}[\![n?(y)\,P]\!]$ has gained access to the high level resource n. Underlying this example is the principle that processes at a given security level σ should have access to resources at security level *at most* σ. We formalize this principle in terms of a relation $P \overset{\Sigma}{\longmapsto} \mathit{err}$, indicating that P violates the security policy Σ.

To prevent such errors, we restrict attention to security policies that are somehow consistent. Let Γ be such a consistent policy; consistency is defined by restricting types so that they respect a subtyping relation. We then introduce a typing system, $\Gamma \vdash P$, which ensures that P can never violate Γ:

> If $\Gamma \vdash P$ then for every context $C[\,]$ such that $\Gamma \vdash C[P]$ and every Q which occurs during the execution of $C[P]$, that is $C[P] \mapsto^{*} Q$, we have $Q \overset{\Gamma}{\not\longmapsto} \mathit{err}$.

Thus our typing system ensures that low level processes will never gain access to high level resources. The typing system implements a particular view of security, which we refer to as the *R-security policy*, as it offers protection to *resources*. Here communication is allowed between high level and low level principals, provided of course that the values involved are at the appropriate security level.

This policy does not rule out the possibility of information leaking indirectly from high security to low security principals. Suppose h is a high channel and hl is a channel with high-level write access and low-level read access in:

$$\mathsf{top}\llbracket h?(x)\,\mathsf{if}\ x = 0\ \mathsf{then}\ \ \mathsf{hl}!\langle 0\rangle\ \ \mathsf{else}\ \ \mathsf{hl}!\langle 1\rangle\ \rrbracket \mid \mathsf{bot}\llbracket\, \mathsf{hl}?(z)\, Q\rrbracket$$

This system can be well-typed although there is some implicit information flow from the high security agent to the low security one; the value received on the high level channel h can be determined by the low level process Q.

It is difficult to formalize exactly what is meant by *implicit information flow* and in the literature various authors have instead relied on *non-interference*, [15, 27, 12, 28], a concept more amenable to formalization, which ensures, at least informally, the absence of implicit information flow.

To obtain such results for the π-calculus we need, as the above example shows, a stricter security policy, which we refer to as the *I-security policy*. This allows a high level principal to read from low level resources but not to write to them. Using the terminology of [2, 8]:

- *write up*: a process at level σ may only write to channels at level σ or above
- *read down*: a process at level σ may only read from channels at level σ or below.

In fact the type checking system remains the same and we only need constrain the notion of type. In this restricted type system well-typing, $\Gamma \Vdash P$, ensures a form of *non-interference*.

To formalize this non-interference result we need to develop a notion of process behaviour, relative to a given security level. Since the behaviour of processes also depends on the type environment in which they operate we need to define a relation

$$P \approx^{\sigma}_{\Gamma} Q$$

which intuitively states that, relative to Γ, there is no observable distinction between the behaviour of P and Q at security level σ; processes running at security level σ can observe no difference in the behaviour of P and Q. Lack of information flow from high to low security levels now means that this relation is invariant under changes in high-level values; or indeed under changes in high-level behaviour.

It turns out that the extent to which this is true depends on the exact formulation of the behavioural equivalence $\approx^{\sigma}_{\Gamma}$. We show that it is *not* true if $\approx^{\sigma}_{\Gamma}$ is based on *observational* equivalence [20] or *must testing* equivalence [22]. But a result can be established if we restrict our attention to *may testing* equivalence (here written $\simeq^{\sigma}_{\Gamma}$). Specifically we will show that, for certain H, K:

If $\Gamma \Vdash^{\sigma} P,\ Q$ and $\Gamma \Vdash^{\mathsf{top}} H,\ K$ then $P \simeq^{\sigma}_{\Gamma} Q$ implies $P \mid H \simeq^{\sigma}_{\Gamma} Q \mid K$.

High-level behaviour can be arbitrarily changed without affecting low-level equivalences. This is the main result of the paper.

The remainder of the paper is organized as follows. In the next section we define the *security π-calculus*, giving a labelled transition semantics and a formal definition of runtime errors. In Section 3 we design a set of types and a typing system which implements the resource control policy. The types are an extension of the IO-types for the π-calculus from [23, 25] in which security levels are associated with specific capabilities. This section also contains Subject Reduction and Type Safety theorems. In Section 4 we motivate the restrictions required on types and terms in order to implement the information control policy. We also give a precise statement of our non-interference result, and give counter-examples to related conjectures based on equivalences other than *may testing*. The proof of our main theorem depends on an analysis of *may testing* in terms of *asynchronous* sequences of actions [7] which in turn depends on detailed operational semantics for our language, where actions are paramterised relative to a typing environment. This is the topic of Section 5, which also contains the proof of our main theorem.

An extended abstract of this paper was originally reported in [17].

2 The Language

The syntax of the *security π-calculus*, given in Figure 1, uses a predefined set of *names*, ranged over by $a, b, \ldots, n$ and a set of *variables*, ranged over by x, y, z. *Identifiers* are either variables or names. *Security annotations*, ranged over by small Greek letters $\sigma, \rho, \ldots$, are taken from a complete lattice $\langle SL, \preceq, \sqcap, \sqcup, \mathsf{top}, \mathsf{bot} \rangle$ of security levels. We also assume for each σ a set of *base values* BV_{σ}, ranged over by *bv*. We require that all syntactic sets be disjoint.

The input construct '$u?(X : \mathrm{A})\, P$' binds all variables in the pattern X while the construct '$(\mathsf{new}\, a : \mathrm{A})\ P$' binds the name a. We have the usual notions of free and bound names and variables, α-equivalence and

FIGURE 1 Syntax

P, Q ::=	*Terms*	X, Y ::=	*Patterns*
$u!\langle v\rangle$	Output	x	Variable
$u?(X:\mathrm{A})\,P$	Input	$(X_1, \ldots, X_k)$	Tuple
$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Matching		
$\sigma[\![P]\!]$	Security level	u, v, w ::=	*Values*
$(\mathsf{new}\, a:\mathrm{A})\ P$	Name creation	bv	Base Value
$P \mid Q$	Composition	a	Name
$*P$	Replication	x	Variable
$\mathbf{0}$	Termination	$(u_1, \ldots, u_k)$	Tuple

substitution. We identify terms up to α-equivalence. Let $\mathsf{fn}(P)$ and $\mathsf{fv}(P)$ denote the set of free names and variables, respectively, of the term P. We use '$P\{\!|^{v}\!/\!_{X}|\!\}$' to denote the substitution of the identifiers occurring in the value v for the variables occurring in the pattern X. For '$P\{\!|^{v}\!/\!_{X}|\!\}$' to be well-defined X and v must have the same structure; to avoid unnecessary complications we assume that a variable can occur at most once in a pattern. The binding constructs have types associated with them; these will be explained in Section 3 but are ignored for the moment. In general these types (and the various security annotations) will be omitted from terms unless they are relevant to the discussion at hand.

The behaviour of a process is determined by the interactions in which it can engage. To define these, we give a labelled transition semantics (LTS) for the language. The set *Act* of *labels*, or *actions*, is defined as follows:

μ ::=	*Actions*
τ	Internal action
$(\tilde{c}:\tilde{\mathrm{C}})a?v$	Input of v on a learning private names $\tilde{c}$
$(\tilde{c}:\tilde{\mathrm{C}})a!v$	Output of v on a revealing private names $\tilde{c}$

In both the input and output actions we require that $\tilde{c}$ be in $\mathsf{fn}(v)$. Let $\mathit{VAct} = \mathit{Act} \setminus \{\tau\}$ be the set of the *visible actions*, either input or output, ranged over by α, β. Whenever these are used we assume that the bound names $\tilde{c}$ occur in the value v. Formally the *bound names* of an action are defined by $\mathsf{bn}(\tau) = \emptyset$ and $\mathsf{bn}((\tilde{c}:\tilde{\mathrm{C}})a!v) = \mathsf{bn}((\tilde{c}:\tilde{\mathrm{C}})a?v) = \{\tilde{c}\}$. We also use $\mathcal{E}(\alpha)$ to denote the bound names in α, together with their types: $\mathcal{E}((\tilde{c}:\tilde{\mathrm{C}})a!v) = \mathcal{E}((\tilde{c}:\tilde{\mathrm{C}})a?v) = (\tilde{c}:\tilde{\mathrm{C}})$. Further, let $\mathsf{n}(\mu)$ be the set of *names* occurring in μ, whether free or bound. We say that the actions '$(\tilde{c}:\tilde{\mathrm{C}})a?v$' and '$(\tilde{c}:\tilde{\mathrm{C}})a!v$' are *complementary*. Given a visible action α, we write $\overline{\alpha}$ to indicate the action complementary to α; note that $\mathsf{bn}(\alpha) = \mathsf{bn}(\overline{\alpha})$ and $\mathcal{E}(\alpha) = \mathcal{E}(\overline{\alpha})$.

FIGURE 2 Labelled Transition Semantics

(L-OUT)

$$\frac{}{a!\langle v\rangle \xrightarrow{a!v} \mathbf{0}}$$

(L-IN)

$$\frac{}{a?(X)\,P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v} P\{\!|{}^{v}\!/\!{}_{X}|\!\}}\quad \tilde{c}\notin \mathsf{fn}(P)$$

(L-OPEN)

$$\frac{P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P'}{(\mathsf{new}\, b:\mathrm{B})\;P \xrightarrow{(b:\mathrm{B})(\tilde{c}:\tilde{\mathrm{C}})a!v} P'}\quad \begin{array}{l} b\neq a\\ b\in \mathsf{fn}(v)\end{array}$$

(L-COM)

$$\frac{P \xrightarrow{\alpha} P',\quad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} (\mathsf{new}\,\mathcal{E}(\alpha))\;(P' \mid Q')}$$

(L-EQ)

$$\frac{}{\mathsf{if}\; u = u \;\mathsf{then}\; P \;\mathsf{else}\; Q \xrightarrow{\tau} P}\qquad \frac{}{\mathsf{if}\; u = w \;\mathsf{then}\; P \;\mathsf{else}\; Q \xrightarrow{\tau} Q}\quad u\neq w$$

(L-CTXT)

$$\frac{P \xrightarrow{\mu} P'}{\begin{array}{l} *P \xrightarrow{\mu} *P \mid P'\\ \sigma[\![P]\!] \xrightarrow{\mu} \sigma[\![P']\!]\end{array}}\qquad \frac{P \xrightarrow{\mu} P'}{\begin{array}{l} P \mid Q \xrightarrow{\mu} P' \mid Q\\ Q \mid P \xrightarrow{\mu} Q \mid P'\end{array}}\quad \mathsf{bn}(\mu)\notin \mathsf{fn}(Q)$$

$$\frac{P \xrightarrow{\mu} P'}{(\mathsf{new}\, a:\mathrm{A})\;P \xrightarrow{\mu} (\mathsf{new}\, a:\mathrm{A})\;P'}\quad a\notin \mathsf{n}(\mu)$$

The LTS is defined in Figure 2 and for the most part the rules are straightforward; it is based on the standard operational semantics from [21], to which the reader is referred for more motivation. Note that in the input rule (L-IN) we are assuming the action $(\tilde{c}:\tilde{\mathrm{C}})a?v$ is well-defined; in principle the process $a?(X)\,P$ can input any value v, but for the action to be valid the bound names $\tilde{c}$ must appear in v and moreover must be new to the process.

Informally a security policy associates with each channel a security level. Our approach, slightly more general, is to incorporate this information into the standard notion of channel types for the π-calculus [23, 25], designed to rule out run-time mistypings, such as sending a triple on a channel designed for pairs. In particular we will associate security levels with *capabilities* on channels, rather than channels themselves, although indirectly we will be able to associate security levels with channels. To this end, *pre-capabilities* and *pre-types* are defined as follows:

cap ::=	*Pre-Capability*
$\mathsf{w}_\sigma\langle \mathrm{A}\rangle$	σ-level process can write values with type A
$\mathsf{r}_\sigma\langle \mathrm{A}\rangle$	σ-level process can read values with type A
A ::=	*Pre-Type*
$\mathbf{B}_\sigma$	Base type
$\{cap_1, \ldots, cap_k\}$	Resource type ($k \geq 0$)
$(\mathrm{A}_1, \ldots, \mathrm{A}_k)$	Tuple type ($k \geq 0$)

We will tend to abbreviate a singleton set of capabilities, $\{cap\}$, to cap.

A *security policy*, Σ, is a finite mapping from names to pre-types. Thus, for example, if Σ maps the channel lh to the pre-type $\{\mathsf{w}_{\mathsf{bot}}\langle \mathrm{B}\rangle,\ \mathsf{r}_{\mathsf{top}}\langle \mathrm{A}\rangle\}$, for some appropriate A, B, then low level processes may write to lh but only high level ones may read from it; this is an approximation of the security associated with a mailbox. On the other hand if Σ maps hl to $\{\mathsf{w}_{\mathsf{top}}\langle \mathrm{B}\rangle,\ \mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle\}$ then hl acts more like an information channel; anybody can read from it but only high level processes may place information there.

The import of a security policy may be underlined by defining what it means to violate it. Our definition is given in Figure 3, in terms of a relation $P \stackrel{\Sigma}{\longmapsto} err$. As an example of runtime errors we have that $\rho[\![a!\langle v\rangle\, P]\!] \stackrel{\Sigma}{\longmapsto} err$ if any of the following hold: (a) $\Sigma(a)$ is undefined, (b) a has no write capability for processes at level ρ, or (c) v contains a base value that is restricted from ρ-level processes. As explained in the Introduction here we are attempting to control access to resources: channels and base values. Principals at level σ have access to all resources at levels up to and including σ. So even if Σ assigns a a low security level $\mathsf{top}[\![a!\langle v\rangle\, P]\!]$ does not cause a runtime error unless v can not be assigned a type appropriate to $\Sigma(a)$.

Example 2.1. Here we assume the policy Σ defined above, mapping lh to $\{\mathsf{w}_{\mathsf{bot}}\langle \mathrm{B}\rangle,\ \mathsf{r}_{\mathsf{top}}\langle \mathrm{A}\rangle\}$ and hl to $\{\mathsf{w}_{\mathsf{top}}\langle \mathrm{B}\rangle,\ \mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle\}$, for some appropriate A, B.

- Consider the process $\mathsf{top}[\![c!\langle \mathsf{hl}\rangle]\!] \mid \mathsf{bot}[\![c?(x)\, x!\langle v\rangle]\!]$. Then after one reduction step there is a security error because $\mathsf{bot}[\![\mathsf{hl}!\langle v\rangle]\!] \stackrel{\Sigma}{\longmapsto} err$. A low security process has write access to security channel hl on which write access is reserved for high-security processes.

- Assuming an appropriate typing for c and v the same security error does not occur in $\mathsf{top}[\![c!\langle \mathsf{lh}\rangle]\!] \mid \mathsf{bot}[\![c?(x)\, x!\langle v\rangle]\!]$. The low security process $\mathsf{bot}[\![\mathsf{lh}!\langle v\rangle\, Q]\!]$ has the right to write on the channel lh.

- If Σ assigns to the channel c a pre-type which includes a capability of the form $\mathsf{r}_{\mathsf{top}}\langle \mathrm{C}\rangle$ then apriori there is no type error in the expression

FIGURE 3 Runtime Errors

$$
\begin{array}{lll}
\text{(E-RD)} & \rho[\![a?(X)\,P]\!] \overset{\Sigma}{\longmapsto} \mathit{err} & \text{if for all A, } \mathsf{r}_\sigma\langle \mathrm{A}\rangle \notin \Sigma(a) \text{ for any } \sigma \preceq \rho \\
\text{(E-WR}_1\text{)} & \rho[\![a!\langle v\rangle]\!] \overset{\Sigma}{\longmapsto} \mathit{err} & \text{if for all A, } \mathsf{w}_\sigma\langle \mathrm{A}\rangle \notin \Sigma(a) \text{ for any } \sigma \preceq \rho \\
\text{(E-WR}_2\text{)} & \rho[\![a!\langle v\rangle]\!] \overset{\Sigma}{\longmapsto} \mathit{err} & \text{if } bv \in \mathbf{B}_\sigma \text{ for some } bv \in v \text{ and } \sigma \not\preceq \rho
\end{array}
$$

$$
\text{(E-STR)} \quad \frac{P \overset{\Sigma}{\longmapsto} \mathit{err}}{P \mid Q \overset{\Sigma}{\longmapsto} \mathit{err}} \quad \frac{P \overset{\Sigma}{\longmapsto} \mathit{err}}{Q \mid P \overset{\Sigma}{\longmapsto} \mathit{err}} \quad \frac{P \overset{\Sigma}{\longmapsto} \mathit{err}}{\rho[\![P]\!] \overset{\Sigma}{\longmapsto} \mathit{err}}
$$

$$
\frac{P \overset{\Sigma, n:\mathrm{A}}{\longmapsto} \mathit{err}}{(\mathsf{new}\, n:\mathrm{A})\ P \overset{\Sigma}{\longmapsto} \mathit{err}}
$$

$c!\langle\mathsf{lh}\rangle$ although intuitively it involves a security leak; a low security agent can read from c a channel which has at least some capability which should only be accessible to high security principals. However it is straightforward to place it in a context in which a security leak occurs: $c!\langle\mathsf{lh}\rangle \mid \mathsf{bot}[\![c?(x)\,x!\langle v\rangle]\!]$. Thus our typing system will also be required to rule out such processes. □

3 Resource Control

Our typing system will apply only to certain security policies, those in which the pre-types are in some sense *consistent*. Consistency is imposed using a system of kinds: the kind RType_σ comprises the value types accessible to processes at security level σ. These kinds are in turn defined using a subtyping relation on pre-capabilities and pre-types.

DEFINITION 3.1. Let <: be the least preorder on pre-capabilities and pre-types such that:

$$
\begin{array}{lrcll}
\text{(U-WR)} & \mathsf{w}_\sigma\langle \mathrm{A}\rangle & <: & \mathsf{w}_\sigma\langle \mathrm{B}\rangle & \text{if B} <: \text{A} \\
\text{(U-RD)} & \mathsf{r}_\sigma\langle \mathrm{A}\rangle & <: & \mathsf{r}_\rho\langle \mathrm{B}\rangle & \text{if A} <: \text{B and } \sigma \preceq \rho \\
\text{(U-BASE)} & \mathbf{B}_\sigma & <: & \mathbf{B}_\rho & \text{if } \sigma \preceq \rho \\
\text{(U-RES)} & \{\mathit{cap}_i\}_{i\in I} & <: & \{\mathit{cap}'_j\}_{j\in J} & \text{if } (\forall j)(\exists i)\ \mathit{cap}_i <: \mathit{cap}'_j \\
\text{(U-TUP)} & (\mathrm{A}_1,\ldots,\mathrm{A}_k) & <: & (\mathrm{B}_1,\ldots,\mathrm{B}_k) & \text{if } (\forall i)\ \mathrm{A}_i <: \mathrm{B}_i
\end{array}
$$

For each ρ, let $RType_\rho$ be the least set that satisfies:

(RT-WR)
$$\frac{\mathrm{A} \in RType_\sigma}{\{\mathsf{w}_\sigma\langle \mathrm{A}\rangle\} \in RType_\rho}\ \sigma \preceq \rho$$

(RT-RD)
$$\frac{\mathrm{A} \in RType_\sigma}{\{\mathsf{r}_\sigma\langle \mathrm{A}\rangle\} \in RType_\rho}\ \sigma \preceq \rho$$

(RT-BASE)
$$\frac{}{\mathbf{B}_\sigma \in RType_\rho}\ \sigma \preceq \rho$$

(RT-WRRD)
$$\frac{\mathrm{A} \in RType_\sigma \quad \mathrm{A}' \in RType_{\sigma'}}{\{\mathsf{w}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{r}_{\sigma'}\langle \mathrm{A}'\rangle\} \in RType_\rho}\ \begin{array}{l}\sigma \preceq \rho\\ \sigma' \preceq \rho\\ \mathrm{A} <: \mathrm{A}'\end{array}$$

(RT-TUP)
$$\frac{\mathrm{A}_i \in RType_\rho \quad (\forall i)}{(\mathrm{A}_1, \ldots, \mathrm{A}_k) \in RType_\rho}$$

Let *RType* be the union of the kinds $RType_\rho$ over all ρ. □

Note that if $\sigma \preceq \rho$ then $RType_\sigma \subseteq RType_\rho$. Intuitively, low level values are accessible to high level processes. However obviously the converse is not true. For example $\mathsf{w}_{\mathsf{top}}\langle\rangle \in RType_{\mathsf{top}}$ but $\mathsf{w}_{\mathsf{top}}\langle\rangle$ is not in $RType_{\mathsf{bot}}$. Note also that there is no relation between subtyping and accessibility at a given security level. For example:

$$\begin{array}{llll}\mathsf{w}_{\mathsf{bot}}\langle\rangle \in RType_{\mathsf{bot}}, & \{\mathsf{w}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{top}}\langle\rangle\} <: \mathsf{w}_{\mathsf{bot}}\langle\rangle & \text{but } \{\mathsf{w}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{top}}\langle\rangle\} & \notin RType_{\mathsf{bot}}\\ \mathsf{r}_{\mathsf{bot}}\langle\rangle \in RType_{\mathsf{bot}}, & \mathsf{r}_{\mathsf{bot}}\langle\rangle <: \mathsf{r}_{\mathsf{top}}\langle\rangle & \text{but } \mathsf{r}_{\mathsf{top}}\langle\rangle & \notin RType_{\mathsf{bot}}\end{array}$$

The compatibility requirement between read and write capabilities in a type (RT-WRRD), in addition to the typing implications discussed in [25], also has security implications. For example suppose $\mathsf{r}_{\mathsf{bot}}\langle \mathbf{B}_\sigma\rangle$ and $\mathsf{w}_{\mathsf{top}}\langle \mathrm{B}\rangle$ are capabilities in a valid channel type, for some type B. Then apriori a high level process can write to the channel while a low level process may read from it. However the only possibility for σ is bot, that is only low level values may be read. Moreover the requirement $\mathrm{B} <: \mathbf{B}_\sigma$ implies that B must also be $\mathbf{B}_{\mathsf{bot}}$. So although high level processes may write to the channel they may only write low level values.

PROPOSITION 3.2. *For every ρ,* RType_ρ *is a preorder with respect to $<:$, with both a partial meet operation $\sqcap$ and a partial join $\sqcup$.* □

Proof. Straightforward adaptation of Proposition 6.2 of [25]. The partial operations $\sqcap$ and $\sqcup$ are first defined by structural induction on types.

FIGURE 4 Typing Rules

(T-ID)
$$\frac{\Gamma(u) <: \mathrm{A}}{\Gamma \vdash u : \mathrm{A}}$$

(T-BASE)
$$\frac{bv \in \mathbf{B}_\sigma}{\Gamma \vdash bv : \mathbf{B}_\sigma}$$

(T-TUP)
$$\frac{\Gamma \vdash v_i : A_i \quad (\forall i)}{\Gamma \vdash (v_1, \ldots, v_k) : (\mathrm{A}_1, \ldots, \mathrm{A}_k)}$$

(T-IN)
$$\frac{\Gamma, X : \mathrm{A} \Vdash^{\sigma} P \qquad \Gamma \vdash u : \mathsf{r}_\sigma\langle \mathrm{A} \rangle}{\Gamma \Vdash^{\sigma} u?(X : \mathrm{A})\, P}$$

(T-OUT)
$$\frac{\Gamma \vdash u : \mathsf{w}_\sigma\langle \mathrm{A} \rangle \qquad \Gamma \vdash v : \mathrm{A}}{\Gamma \Vdash^{\sigma} u!\langle v \rangle}$$

(T-EQ)
$$\frac{\Gamma \vdash u : \mathrm{A}, v : \mathrm{B} \qquad \Gamma \Vdash^{\sigma} Q \qquad \Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \Vdash^{\sigma} P}{\Gamma \Vdash^{\sigma} \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

(T-SR)
$$\frac{\Gamma \Vdash^{\sigma \sqcap \rho} P}{\Gamma \Vdash^{\sigma} \rho[\![P]\!]}$$

(T-NEW)
$$\frac{\Gamma, a : \mathrm{A} \Vdash^{\sigma} P}{\Gamma \Vdash^{\sigma} (\mathsf{new}\, a : \mathrm{A})\ P}$$

(T-STR)
$$\frac{\Gamma \Vdash^{\sigma} P,\ Q}{\Gamma \Vdash^{\sigma} P \mid Q,\ *P,\ \mathbf{0}}$$

Typical clauses are

$$\begin{aligned}
\mathsf{r}_\sigma\langle \mathrm{A} \rangle \sqcap \mathsf{r}_{\sigma'}\langle \mathrm{A}' \rangle &= \mathsf{r}_{\sigma \sqcap \sigma'}\langle \mathrm{A} \sqcap \mathrm{A}' \rangle \\
\mathsf{w}_\sigma\langle \mathrm{A} \rangle \sqcap \mathsf{w}_{\sigma}\langle \mathrm{A}' \rangle &= \mathsf{w}_{\sigma}\langle \mathrm{A} \sqcup \mathrm{A}' \rangle \\
\mathsf{r}_\sigma\langle \mathrm{A} \rangle \sqcup \mathsf{r}_{\sigma'}\langle \mathrm{A}' \rangle &= \mathsf{r}_{\sigma \sqcup \sigma'}\langle \mathrm{A} \sqcup \mathrm{A}' \rangle \\
\mathsf{w}_\sigma\langle \mathrm{A} \rangle \sqcup \mathsf{w}_{\sigma}\langle \mathrm{A}' \rangle &= \mathsf{w}_{\sigma}\langle \mathrm{A} \sqcap \mathrm{A}' \rangle
\end{aligned}$$

One can then show, by induction on the definitions, that:

> $\mathrm{A} \in \mathit{RType}_\rho$ and $\mathrm{A} \in \mathit{RType}_{\rho'}$ implies $\mathrm{A} \sqcap \mathrm{B} \in \mathit{RType}_{\rho \sqcap \rho'}$ and $\mathrm{A} \sqcup \mathrm{B} \in \mathit{RType}_{\rho \sqcup \rho'}$.

Finally it is straightforward to show that $\sqcap$ and $\sqcup$, defined in this manner, are indeed partial meet and partial join operators. □

We now discuss the typing system, which is defined using restricted security policies, called type environments. A *type environment* is a finite mapping from identifiers (names and variables) to types. We adopt some standard notation. For example, let '$\Gamma, u : \mathrm{A}$' denote the obvious extension of Γ; '$\Gamma, u : \mathrm{A}$' is only defined if u is not in the domain of Γ. The subtyping relation $<:$ together with the partial operators $\sqcap$ and $\sqcup$ may also be extended to environments. For example $\Gamma <: \Delta$ if for all u in the domain of Δ, $\Gamma(u) <: \Delta(u)$. The partial meet enables us to define more subtle extensions. For example $\Gamma \sqcap \{u : \mathrm{A}\}$ may be defined even if u is already in the domain of Γ. It is well defined when $\Gamma(u) \sqcap \mathrm{A}$ exists, in which case it maps u to this type. We will normally abbreviate the simple environment $\{u : \mathrm{A}\}$ to $u : \mathrm{A}$ and moreover use $v : \mathrm{A}$ to denote its obvious generalisation to values; this is only well-defined when the value v has the same structure as the type A.

The typing system is given in Figure 4 where the judgements are of the form '$\Gamma \Vdash^{\sigma} P$'. If $\Gamma \Vdash^{\sigma} P$ we say that P is a *σ-level process*. Also, let '$\Gamma \vdash P$' abbreviate '$\Gamma \Vdash^{\mathsf{top}} P$'.

Intuitively '$\Gamma \Vdash^{\sigma} P$' indicates that the process P will not cause any security errors if executed with security clearance σ. The rules are very similar to those used in papers such as [23, 25] for the standard IO typing of the π-calculus. The rule (T-EQ), a non-standard rule for matching, is taken from [25], where it is explained and motivated. Essentially it allows processes to accumulate type information on names, information which might be received piecemeal on different occurrences of the same name. It is shown to be particularly useful in types systems, such as ours, in which types are viewed as *capabilities*.

The only significant use of the security levels is in the (T-IN) and (T-OUT) rules, where the channels are required to have a specific security level. This is inferred using auxiliary value judgements, of the form $\Gamma \vdash v : \mathrm{A}$. It is interesting to note that security levels play no direct role in their derivation. One might expect that the judgements for values would need to ensure that a value written to a channel be accessible at the appropriate security level. This job, however, is already handled by our definition of types. For example, in order for $\mathsf{w}_{\sigma}\langle \mathrm{A} \rangle$ to be a type, A must be a type accessible to σ.

The typing system enjoys many expected properties, the proof of which we leave to the reader.

PROPOSITION 3.3.

- (SPECIALIZATION) $\Gamma \vdash v : \mathrm{A}$ *and* $\mathrm{A} <: \mathrm{B}$ *then* $\Gamma \vdash v : \mathrm{B}$
- (WEAKENING) $\Gamma \Vdash^{\sigma} P$ *and* $\Delta <: \Gamma$ *then* $\Delta \Vdash^{\sigma} P$
- (RESTRICTION) $\Gamma, u : \mathrm{A} \Vdash^{\sigma} P$ *and* $u \notin \mathsf{fv}(P) \cup \mathsf{fn}(P)$ *implies* $\Gamma \Vdash^{\sigma} P$. □

The main technical tool required for Subject Reduction is, as usual, a substitution result.

LEMMA 3.4 (SUBSTITUTION). *If* $\Gamma \vdash v : \mathrm{A}$ *then*

- $\Gamma \vdash u : \mathrm{A}$ *implies* $\Gamma \vdash u\{\!\!\{^{v}/_{X}\}\!\!\}$
- $\Gamma, X : \mathrm{A} \Vdash^{\sigma} P$ *implies* $\Gamma \Vdash^{\sigma} P\{\!\!\{^{v}/_{X}\}\!\!\}$

□

Proof. Easily reconstructed from the corresponding proof in [25], Lemma 4.7. □

THEOREM 3.5 (SUBJECT REDUCTION). *Suppose* $\Gamma \vDash^{\sigma} P$. *Then*

- $P \xrightarrow{\tau} Q$ *implies* $\Gamma \vDash^{\sigma} Q$
- $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v} Q$ *implies there exists a type* A *such that* $\Gamma \vdash a : \mathsf{r}_{\delta}\langle \mathrm{A} \rangle$ *for some* $\delta \preceq \sigma$, *and if* $\Gamma \sqcap v : \mathrm{A}$ *is well-defined then* $\Gamma \sqcap v : \mathrm{A} \vDash^{\sigma} Q$.
- $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$ *implies there exists a type* A *such that* $\Gamma \vdash a : \mathsf{w}_{\delta}\langle \mathrm{A} \rangle$ *for some* $\delta \preceq \sigma$, $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \mathrm{A}$ *and* $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \vDash^{\sigma} Q$.

□

Proof. The three statements are proved simultaneously by induction on the inference $P \xrightarrow{\mu} Q$. We examine some cases.

The rule (L-IN): $a?(X : \mathrm{A})\, P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v} P\{\!\!\{ v/X \}\!\!\}$. Because $\Gamma \vDash^{\sigma} a?(X : \mathrm{A})\, P$ we know $\Gamma \vdash a : \mathsf{r}_{\sigma}\langle \mathrm{A} \rangle$ and $\Gamma, X : \mathrm{A} \vDash^{\sigma} P$. Now suppose $\Gamma \sqcap v : \mathrm{A}$ is well-defined. By Weakening we obtain $(\Gamma \sqcap v : \mathrm{A}), X : \mathrm{A} \vDash^{\sigma} P$ and therefore applying the Substitution Lemma we obtain $\Gamma \sqcap v : \mathrm{A} \vDash^{\sigma} P\{\!\!\{ v/X \}\!\!\}$. The rule (L-OUT) is similar.

We consider one example of the rule (L-CTXT): $\rho[\![P]\!] \xrightarrow{\mu} \rho[\![P']\!]$ because $P \xrightarrow{\mu} P'$. The precise details depend on μ, but in each of the three possibilities the reasoning is very similar; so suppose μ is an input action $(\tilde{c}:\tilde{\mathrm{C}})a?v$. We know, by well-typing, that $\Gamma \vDash^{\sigma \sqcap \rho} P$ and therefore we may apply induction to obtain a type A and a $\delta \preceq \sigma \sqcap \rho$ such that $\Gamma \vdash a : \mathsf{r}_{\delta}\langle \mathrm{A} \rangle$; in particular $\delta \preceq \sigma$. Now suppose $\Gamma \sqcap v : \mathrm{A}$ exists. Then, again by induction, we know $\Gamma \sqcap v : \mathrm{A} \vDash^{\sigma \sqcap \rho} P'$ and therefore applying the typing rule (T-SR) we obtain the required $\Gamma \sqcap v : \mathrm{A} \vDash^{\sigma} \rho[\![P']\!]$.

The rule (L-OPEN): $(\mathsf{new}\, b : \mathrm{B})\ P \xrightarrow{(b:\mathrm{B})(\tilde{c}:\tilde{\mathrm{C}})a!v} P'$ because $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P'$. Here we know $\Gamma,\ b : \mathrm{B} \vDash^{\sigma} P$ and therefore applying induction to the action $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P'$ we obtain a type A such that $\Gamma,\ b : \mathrm{B}, \tilde{c} : \tilde{\mathrm{C}} \vDash^{\sigma} P'$ and $\Gamma,\ b : \mathrm{B}, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \mathrm{A}$; moreover $\Gamma,\ b : \mathrm{B} \vdash a : \mathsf{r}_{\delta}\langle \mathrm{A} \rangle$, for some $\delta \preceq \sigma$. However since (L-OPEN) requires that $b \neq a$ we may conclude, as required, $\Gamma \vdash a : \mathsf{r}_{\delta}\langle \mathrm{A} \rangle$.

As a final example consider the rule (L-COM): $P | Q \xrightarrow{\tau} (\mathsf{new}\, \mathcal{E}(\alpha))\ (P' | Q')$ because $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{\overline{\alpha}} Q'$. Without loss of generality we may assume α is the input action $(\tilde{c}:\tilde{\mathrm{C}})a?v$. We know $\Gamma \vDash^{\sigma} P,\ Q$ and therefore we may apply induction to both reduction statements. Applying it to $Q \xrightarrow{\overline{\alpha}} Q'$ we obtain $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \mathrm{A}$ and $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \vDash^{\sigma} Q$. The former implies that $\Gamma \sqcap v : \mathrm{A}$ is well-defined and therefore induction applied to $P \xrightarrow{\alpha} P'$ gives $\Gamma \sqcap v : \mathrm{A} \vDash^{\sigma} P'$. Since $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \mathrm{A}$ it follows that $\Gamma, \tilde{c} : \tilde{\mathrm{C}} <: \Gamma \sqcap v : \mathrm{A}$ and therefore by Weakening we have $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \vDash^{\sigma} P'$. An application of (T-STR), followed by (T-NEW), gives the required $\Gamma \vDash^{\sigma} (\mathsf{new}\, \mathcal{E}(\alpha))\ (P' \mid Q')$.

□

Remark. Most of the restrictions imposed on types are essential to achieving Subject Reduction, but a few are not. First, the Subject Reduction theorem remains true if we weaken (U-WR) to:

$$\mathsf{w}_\sigma\langle \mathrm{A}\rangle <: \mathsf{w}_\rho\langle \mathrm{B}\rangle \quad \text{if } \mathrm{B} <: \mathrm{A} \text{ and } \sigma \preceq \rho,$$

and the proof remains the same. Were we to adopt this rule, it would be true that every process typable at level σ would also be typable at level ρ, for $\sigma \preceq \rho$. Given our actual definition, this is not true. Nonetheless, every process typable at σ can be trivially rewritten so that it is typable at ρ given our definition (one must simply surround output actions with explicit security restrictions). We have adopted the stronger rule because it is necessary in the next section and results in no substantive loss of expressivity. Note also that with our more restrictive defintion of type the statement of the output clause of Subject Reduction can be strengthened; because of the restriction the only possible value of δ is σ. However the clause as it stands remains true with the weakened version of (U-WR).

Second, we have limited types to contain at most one read and one write capability. We have done so to simplify the proofs, particularly in the next section. This clearly results in a loss of expressiveness. We have yet to find, however, a compelling example that requires a resource to have more than one read or one write capability. It is usually sensible to simply take the meet. □

We can now prove the first main result:

THEOREM 3.6 (TYPE SAFETY). *If* $\Gamma \vdash P$ *then for every closed context* $C[\,]$ *such that* $\Gamma \vdash C[P]$ *and every* Q *such that* $C[P] \xrightarrow{\tau}{}^{*} Q$ *we have* $Q \not\xmapsto{\Gamma} \mathrm{err}$ □

Proof. By Subject Reduction we know that $\Gamma \vDash^{\mathsf{top}} Q$ and therefore it is sufficient to prove that $\Gamma \vDash^{\mathsf{top}} Q$ implies $Q \not\xmapsto{\Gamma} err$. In fact we prove the contrapositive, $Q \xmapsto{\Gamma} err$ implies $\Gamma \not\vDash^{\mathsf{top}} Q$ by induction on the definition of $Q \xmapsto{\Gamma} err$.

This is a straightforward inductive proof on the derivation of $Q \xmapsto{\Gamma} err$. For example consider the case (E-RD). Suppose that $\rho[\![a?(X)\,P]\!] \xmapsto{\Gamma} err$ because $\sigma \preceq \rho$ implies for all A, $\mathsf{r}_\sigma\langle \mathrm{A}\rangle \notin \Sigma(a)$. By supposition, we have that $\Gamma(a)$ either has no read capability or it has a read capability at level δ, where $\delta \not\preceq \rho$. In either case, the judgement $\Gamma \vDash^{\rho} a?(X)\,P$ cannot be derived, and therefore $\Gamma \vDash^{\mathsf{top}} \rho[\![a?(X)\,P]\!]$ is also underivable. □

We end this section with a brief discussion on the use of the syntax $\sigma[\![P]\!]$ in our language. We have primarily introduced it in order to discuss typing issues. Having defined our typing system we may now view $\sigma[\![P]\!]$

simply as notation for the fact that, relative to the current typing environment Γ, the process P is well-typed at level σ, i.e. $\Gamma \models^{\sigma} P$. Technically we can view $\sigma[\![P]\!]$ to be *structurally equivalent* to P, assuming we are working in an environment Γ such that $\Gamma \models^{\sigma} P$. This will be formalised in Section 5.

4 Information Flow

We have shown in the previous sections that, in well-typed systems, processes running at a given security level can only access resources appropriate to that level. However, as pointed out in the Introduction this does not rule out (implicit) information flow between levels. Consider the following system

$$\mathsf{top}[\![h?(x)\,\mathsf{if}\ x = 0\ \mathsf{then}\ \ \mathsf{hl}!\langle 0\rangle\ \ \mathsf{else}\ \ \mathsf{hl}!\langle 1\rangle\,]\!]\ \mid \mathsf{bot}[\![\,\mathsf{hl}?(z)\,Q]\!] \qquad (\star)$$

executing in an environment in which h is a top-level read/write channel and hl is a top-level write and bot-level read channel. This system can be well-typed, using *R-types*, so the processes only access resources appropriate to their security level. Nevertheless there is some *implicit* flow of information from top to bot; the low-level process, $\mathsf{bot}[\![\mathsf{hl}?(z)\,Q]\!]$, by testing the value received on z can gain some information about the high-level value x received by the high-level process on the high-level channel h.

One way of formalizing this notion of flow of information is to consider the behaviour of processes and how it can be influenced. If the behaviour of low-level processes is independent of any high-level values in its environment then we can say that there can be no implicit flow of information from high-level to low-level. This is *not* the case in the example above. Suppose, for example, that Q is the code fragment '$\mathsf{if}\ z = 0\ \mathsf{then}\ l_1!\langle\rangle\ \ \mathsf{else}\ l_2!\langle\rangle$'. If $(\star)$ were placed in an environment with '$\mathsf{top}[\![h!\langle 0\rangle]\!]$', then the resource l_1 would be called. If, instead, $(\star)$ were placed in an environment with '$\mathsf{top}[\![h!\langle 42\rangle]\!]$', then l_2 would be called. In other words the behaviour of the low-level process can be influenced by high-level changes; there is a possibility of information flow downwards.

This is not surprising in view of the type associated with the channel hl; in the terminology of [2] it allows a *write down* from a high-level process to a low-level process. Thus if we are to eliminate implicit information flow between levels in well-typed processes we need to restrict further the allowed types; types such as $\{\mathsf{w}_{\mathsf{top}}\langle\rangle,\ \mathsf{r}_{\mathsf{bot}}\langle\rangle\}$ clearly contradict the spirit of secrecy. Thus, for the rest of the paper we work with the more restrictive set *IType*, the *Information types*. In order for $\{\mathsf{w}_{\sigma}\langle \mathrm{A}\rangle,\ \mathsf{r}_{\sigma'}\langle \mathrm{A}'\rangle\}$ to be in *IType*, it must be that $\sigma \preceq \sigma'$; this is not necessarily true for types in *RType*.

DEFINITION 4.1. For each ρ, let $IType_\rho$, be the least set that satisfies the rules in Definition 3.1, with (RT-WRRD) replaced by:

$$\text{(IT-WRRD)}\quad \frac{\begin{array}{l}\mathrm{A} \in IType_\sigma \\ \mathrm{A}' \in IType_{\sigma'}\end{array}}{\{\mathsf{w}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{r}_{\sigma'}\langle \mathrm{A}'\rangle\} \in IType_\rho}\quad \begin{array}{l}\sigma \preceq \sigma' \\ \sigma' \preceq \rho \\ \mathrm{A} <: \mathrm{A}'\end{array}$$

Let *IType* be the union of $IType_\rho$ over all ρ. We write $\Gamma \Vdash^{\sigma} P$ if $\Gamma \vdash^{\sigma} P$ can be derived from the rules of Figure 4 using these more restrictive types. □

All of the results of the previous section carry over to the stronger typing system; we leave their elaboration to the reader.

Unfortunately, due to the expressiveness of our language, the use of *I-types* still does not preclude information flow downwards, between levels. Consider the system

$$\mathsf{top}\llbracket h?(x)\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{bot}\llbracket l!\langle 0\rangle\rrbracket\ \mathsf{else}\ \mathsf{bot}\llbracket l!\langle 1\rangle\rrbracket\rrbracket \mid \mathsf{bot}\llbracket l?(z)\,Q\rrbracket$$

executing in an environment in which h is a top-level read/write channel and l is a bot-level read/write channel. This system can be well-typed using *I-types*, but there still appears to be some some implicit flow of information from top to bot. The problem here is that our syntax allows a high-level process, which can not write to low-level channels, to evolve into a low-level process which does have this capability; we need to place a boundary between low- and high-level processes which ensures a high-level process never gains write access to low-level channels. This is the aim of the following definition:

DEFINITION 4.2. Define the security levels of a term below ρ, $\mathrm{sl}_\rho(P)$, as follows:

$$\begin{array}{rrr}
\mathrm{sl}_\rho(*P) = \mathrm{sl}_\rho(P) & \mathrm{sl}_\rho(\mathbf{0}) = \{\rho\} & \mathrm{sl}_\rho(\sigma\llbracket P\rrbracket) = \{\sigma \sqcap \rho\} \cup \mathrm{sl}_{\sigma\sqcap\rho}(P) \\
\mathrm{sl}_\rho((\mathsf{new}\, a\!:\!\mathrm{A})\ \ P) = \mathrm{sl}_\rho(P) & \mathrm{sl}_\rho(u!\langle v\rangle) = \{\rho\} & \mathrm{sl}_\rho(P \mid Q) = \mathrm{sl}_\rho(P) \cup \mathrm{sl}_\rho(Q) \\
\mathrm{sl}_\rho(u?(X\!:\!\mathrm{B})\,P) = \mathrm{sl}_\rho(P) & \multicolumn{2}{r}{\mathrm{sl}_\rho(\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q) = \mathrm{sl}_\rho(P) \cup \mathrm{sl}_\rho(Q)}
\end{array}$$

A process P is *σ-free* if for every ρ in $\mathrm{sl}_{\mathsf{top}}(P)$, $\rho \not\preceq \sigma$. □

Note that $\mathsf{top} \in \mathrm{sl}_{\mathsf{top}}(P)$ for every P and therefore if P is *σ-free* it must be that $\sigma \neq \mathsf{top}$.

In general σ-freedom restricts the ability of processes to reduce their security level to σ; this will restrict their ability to write to σ-level processes, but not their ability to read from them. The definition may appear complicated but unfortunately it is not sufficient to disallow occurrences of $\sigma[\![\ ]\!]$ from P. Consider for example the process $\rho_1[\![\rho_2[\![Q]\!]]\!]$, where $\rho_1 \not\preceq \sigma$. This does not contain any occurrence of $\sigma[\![\ ]\!]$, (assuming it does not occur in Q), but if $\rho_1 \sqcap \rho_2 = \sigma$ then effectively Q is running at security level σ, although there is no occurrence of σ in the term.

To what extent, therefore, does σ-freedom preclude implicit information flow? We avoid giving a formal definition of *implicit information flow*. Instead we can demand that, in order to informally preclude such information flow, low-level behaviour be completely independent of arbitrary high-level behaviour; it should not be possible to influence low-level behaviour by changing high-level behaviour. This can be formalized as a non-interference result of the form:

> Suppose P and Q are σ-level processes and $P \approx^\sigma Q$. Further suppose that H and K are arbitrary top-level σ-free processes. Then $P \mid H \approx^\sigma Q \mid K$.

Here $\approx^\sigma$ is some form of behavioural equivalence that is sensitive only to behaviour of processes that are σ-level or lower. It turns out that such a result is very dependent on the exact formulation used, as the following example illustrates.

Let A denote the type $\{\mathsf{w}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{bot}}\langle\rangle\}$ and B denote $\{\mathsf{r}_{\mathsf{bot}}\langle\rangle\}$. Further, let Γ map a and b to A and B, respectively, and n to the type $\{\mathsf{w}_{\mathsf{bot}}\langle \mathrm{A}\rangle, \mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle\}$. Now consider the terms P and H defined by

$$P \Leftarrow \mathsf{bot}[\![n!\langle a\rangle \mid n?(x:\mathrm{A})\ x!\langle\rangle]\!] \qquad H \Leftarrow \mathsf{top}[\![n?(x:\mathrm{B})\ b?(y)\,\mathbf{0}]\!]$$

It is very easy to check that $\Gamma \Vdash P, H$ and that H is bot-free. Note that in the term $P|H$ there is contention between the low and high-level processes for who will receive a value on the channel n. This means that if we were to base the semantic relation $\approx$ on any of *strong bisimulation equivalence*, *weak bisimulation equivalence*, [20], or *must testing*, [22], we would have

$$P \mid \mathbf{0} \not\approx^\sigma P \mid H$$

The essential reason is that the consumption of writes can be detected; the reduction

$$P \mid H \xrightarrow{\tau} \mathsf{bot}[\![n?(x:\mathrm{A})\ x!\langle\rangle]\!] \mid \mathsf{top}[\![b?(y)\,.\,\mathbf{0}]\!]$$

cannot be matched by $P|\mathbf{0}$. Using the terminology of [22], $P|\mathbf{0}$ *guarantees* the test $\mathsf{bot}[\![a?(x)\,\omega!\langle\rangle]\!]$ whereas $P \mid H$ does not.

Even obtaining results with respect to *may testing*, defined in Section 5, is delicate. If we allowed *synchronous* tests then we would also have:

$$P \mid \mathbf{0} \not\approx^\sigma P \mid H$$

Let T be the test $\mathsf{bot}[\![b!\langle\rangle\,\omega!\langle\rangle]\!]$. Then $P \mid H \mid T$ may eventually produce an output on ω whereas $P \mid \mathbf{0} \mid T$ cannot. However, since our language is asynchronous, such tests are not allowed.

In the following section, we prove a non-interference result using may testing on processes typable using *I-types*. Note that, as indicated by the examples above, we can rewrite our informal notion of non-interference in an equivalent, but simpler manner. It is sufficient to insist that

$P \,|\, H \approx^\sigma P$ for all σ-level processes P and top-level σ-free processes H.

This is the formulation used in [10], and if the equivalence $\approx^\sigma$ enjoys reasonable properties this is sufficient to ensure

$$P \mid H \approx^\sigma P \mid K$$

for all top-level σ-free processes H, K. Indeed our proof will proceed in this manner.

5 Noninterference up to May Testing

May equivalence is defined in terms of tests. A *test* is a process with an occurrence of a new reserved resource name ω. We use T to range over tests, with the typing rule $\Gamma \Vdash^\sigma \omega!\langle\rangle$ for all Γ and σ. When placed in parallel with a process P, a test may interact with P, producing an output on ω if some desired behaviour of P has been observed.

DEFINITION 5.1. We write $T\Downarrow$ if $T \xrightarrow{\tau}{}^* T' \xrightarrow{\omega!\langle\rangle}$. □

We wish to capture the behaviour of processes at a given level of security. Consequently we only compare their ability to pass tests that are well-typed at that level. The definition must also take into account the environment in which the processes are used, as this determines the security level associated with resources.

DEFINITION 5.2. We write $P \simeq^\sigma_\Gamma Q$ if for every test T such that $\Gamma \Vdash^\sigma T$:

$$(P \mid T)\Downarrow \text{ if and only if } (Q \mid T)\Downarrow$$ □

Note that in the definition of '$P \simeq^\sigma_\Gamma Q$', P and Q need not be well-typed. Γ is a constraint on the environment in which the processes are run, not on the processes themselves. Nevertheless, at least in this paper, the definition will only be applied to processes which are well-behaved with respect to the constraint Γ.

We can now state the main result of the paper.

FIGURE 5 Context LTS

(C-RED)
$$\frac{P \xrightarrow{\tau} P'}{\Delta \triangleright P \xrightarrow{\tau}_\sigma \Delta \triangleright P'}$$

(C-OUT)
$$\frac{\Delta \Vdash a : \mathsf{r}_\delta\langle \mathrm{B}\rangle}{\Delta \triangleright a!\langle v\rangle \xrightarrow{a!v}_\sigma \Delta \triangleright \mathbf{0}} \quad \delta \preceq \sigma$$

(C-IN)
$$\frac{\Delta \Vdash a : \mathsf{w}_\delta\langle \mathrm{B}\rangle \qquad \Delta, \tilde{c} : \tilde{\mathrm{C}} \Vdash v : \mathrm{B}}{\Delta \triangleright a?(X:\mathrm{B})\, P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v}_\sigma \Delta, \tilde{c} : \tilde{\mathrm{C}} \triangleright P\{\!|{}^{v}/_{X}|\!\}} \quad \begin{array}{l}\delta \preceq \sigma \\ \tilde{c} \notin \mathsf{fn}(P)\end{array}$$

(C-OPEN)
$$\frac{\Delta \triangleright P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v}_\sigma \Delta' \triangleright P'}{\Delta \triangleright (\mathsf{new}\, b : \mathrm{B})\ P \xrightarrow{(b:\mathrm{B})(\tilde{c}:\tilde{\mathrm{C}})a!v}_\sigma \Delta' \triangleright P'} \quad \begin{array}{l} b \neq a \\ b \in \mathsf{fn}(v)\end{array}$$

(C-CTXT)
$$\frac{\Delta \triangleright P \xrightarrow{\mu}_\sigma \Delta' \triangleright P'}{\begin{array}{l}\Delta \triangleright *P \xrightarrow{\mu}_\sigma \Delta' \triangleright *P \mid P' \\ \Delta \triangleright \rho[\![P]\!] \xrightarrow{\mu}_\sigma \Delta' \triangleright \rho[\![P']\!]\end{array}}$$

$$\frac{\Delta \triangleright P \xrightarrow{\mu}_\sigma \Delta' \triangleright P'}{\begin{array}{l}\Delta \triangleright P \mid Q \xrightarrow{\mu}_\sigma \Delta' \triangleright P' \mid Q \\ \Delta \triangleright Q \mid P \xrightarrow{\mu}_\sigma \Delta' \triangleright Q \mid P'\end{array}} \quad \mathsf{bn}(\mu) \notin \mathsf{fn}(Q)$$

$$\frac{\Delta \triangleright P \xrightarrow{\mu}_\sigma \Delta' \triangleright P'}{\Delta \triangleright (\mathsf{new}\, a : \mathrm{A})\ P \xrightarrow{\mu}_\sigma \Delta' \triangleright (\mathsf{new}\, a : \mathrm{A})\ P'} \quad a \notin \mathsf{n}(\mu)$$

THEOREM 5.3 (NON-INTERFERENCE). *If* $\Gamma \Vdash^{\sigma} P, Q$ *and* $\Gamma \Vdash^{\mathsf{top}} H, K$ *where* H *and* K *are* σ*-free processes, then* $P \simeq^{\sigma}_{\Gamma} Q$ *implies* $P \mid H \simeq^{\sigma}_{\Gamma} Q \mid K$.

As already indicated this theorem will follow if we can establish that

$$P \simeq^{\sigma}_{\Gamma} P \mid H$$

for all P, H satisfying the constraints of the theorem. The proof of this fact relies on constructing sufficient conditions to guarantee that two processes are *may equivalent*. This is the topic of the next subsection, which is followed by a subsection giving the proof of the non-interference result.

5.1 Sufficient Conditions

The purpose of the LTS semantics given in Figure 5 is to capture the possible interactions in which a process can engage with its environment. However our language is typed and therefore the type environment, constraining the environment, may forbid interactions which the process, in principle, is capable of performing. For example if Γ is an environment which associates with the channel a only a read capability then we will have the identity

$$a?(X)\,P \simeq^{\sigma}_{\Gamma} \mathbf{0}$$

because there can be no test T such that $\Gamma \Vdash^{\sigma} T$ which can interact with $a?(X)\,P$ to discover its behaviour.

In other words we need to modify the LTS semantics to take into account the environment in which the process is being tested. This leads us to judgements of the form $\Gamma \triangleright P \xrightarrow{\mu}_{\sigma} \Gamma' \triangleright P'$. Intuitively, this should be read:

> Let T be a test such that $\Gamma \Vdash^{\sigma} T$. Then P can interact with T by performing the action μ and evolving to P'. As a result of this interaction, the capabilities of the context may be increased, as reflected in Γ'.

The modified LTS is defined in Figure 5 and the rules are straightforward. However note that in the rule (C-OUT) it should be informally understood that the environment already knows the value v being output; it is only in the rule (C-OPEN) where the environment learns new information.

Some properties of this modified LTS are easy to establish. For example in $\Gamma \triangleright P \xrightarrow{\mu}_{\sigma} \Gamma' \triangleright P'$ the new environment Γ' is completely determined by Γ and the action μ. If μ is τ then Γ' coincides with Γ; otherwise it is Γ augmented with the type environment $\mathcal{E}(\mu)$, the bound names together with their declared types. For this reason the following Lemma is easily established:

LEMMA 5.4. *$\Gamma \triangleright P \xrightarrow{\mu}_{\sigma} \Gamma' \triangleright P'$ and $\Gamma \Vdash P$ implies $\Gamma' \Vdash P'$.* □

Proof. By induction on the derivation of the judgement $\Gamma \triangleright P \xrightarrow{\mu}_{\sigma} \Gamma' \triangleright P'$. □

There are also very simple conditions which ensure that apriori untyped actions may be performed in a type environment:

LEMMA 5.5. *Let $P \xrightarrow{\alpha} Q$.*

- *Suppose α is $(\tilde{c}:\tilde{\mathrm{C}})a?v$. If $\Gamma \Vdash a:\mathsf{w}_{\delta}\langle \mathrm{B}\rangle$, where $\delta \preceq \sigma$, and $\Gamma,\ \tilde{c}:\tilde{\mathrm{C}} \Vdash v:\mathrm{B}$ then $\Gamma \triangleright P \xrightarrow{\alpha}_{\sigma} \Gamma,\ \tilde{c}:\tilde{\mathrm{C}} \triangleright Q$.*
- *Suppose α is $(\tilde{c}:\tilde{\mathrm{C}})a!v$. If $\Gamma \Vdash a:\mathsf{r}_{\delta}\langle \mathrm{B}\rangle$, where $\delta \preceq \sigma$, then $\Gamma \triangleright P \xrightarrow{\alpha}_{\sigma} \Gamma,\ \tilde{c}:\tilde{\mathrm{C}} \triangleright Q$.*

□

Proof. A simple proof by induction on the derivation of $P \xrightarrow{\alpha} Q$. □

However it is the following Decomposition Lemma which makes the augmented LTS of interest:

LEMMA 5.6 (DECOMPOSITION). *Suppose* $\Gamma \Vdash^{\sigma} T$ *and* $\Gamma \Vdash P$. *Then* $P \mid T \xrightarrow{\tau} R$ *implies one of the following:*

(a) $R = P' \mid T$ *and* $\Gamma \triangleright P \xrightarrow{\tau} \Gamma \triangleright P'$,
(b) $R = P \mid T'$ *and* $T \xrightarrow{\tau} T'$,
(c) $R = (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}})\ P' \mid T'$ *and* $\Gamma \triangleright P \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a!v}_{\sigma} \Gamma' \triangleright P'$ *and* $T \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a?v} T'$, *or*
(d) $R = (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}})\ P' \mid T'$ *and* $\Gamma \triangleright P \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a?v}_{\sigma} \Gamma' \triangleright P'$ *and* $T \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a!v} T'$.

Furthermore in the last two cases $\Gamma' \Vdash^{\sigma} T'$. □

Proof. By induction on the derivation of $P \mid T \xrightarrow{\tau} R$. The only interesting case is when this is inferred using the rule (L-COM), where R has the form $(\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}})\ (P' \mid T')$. There are two possibilities.

First suppose $P \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a?v} P'$, $T \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a!v} T'$. By Subject Reduction applied to $\Gamma \Vdash^{\sigma} T$ we know $\Gamma \Vdash a : \mathsf{w}_{\delta}\langle \mathrm{B} \rangle$, for some $\delta \preceq \sigma$ and some type B such that $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \Vdash v : \mathrm{B}$. We may now apply the previous Lemma, to obtain the required $\Gamma \triangleright P \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a?v}_{\sigma} \Gamma, \tilde{c} : \tilde{\mathrm{C}} \triangleright P'$. The fact that $\Gamma' \Vdash^{\sigma} T'$ follows by Subject Reduction.

The second case, when P outputs and T inputs, is similar. Here $P \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a!v} P'$, $T \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a?v} T'$ and the only difficulty is to show that $\Gamma, \tilde{c} : \tilde{C} \Vdash^{\sigma} T'$. We know, by Subject Reduction, that $\Gamma \Vdash a : \mathsf{r}_{\sigma}\langle \mathrm{A} \rangle$ and if $\Gamma \sqcap v : \mathrm{A}$ exists then $\Gamma \sqcap v : \mathrm{A} \Vdash^{\sigma} T'$. However we also know $\Gamma \Vdash P$ and therefore by Subject Reduction, applied to $P \xrightarrow{(\tilde{c} : \tilde{\mathrm{C}})a!v} P'$ we know $\Gamma, \tilde{c} : \tilde{C} \Vdash v : \mathrm{B}$ for some type B such that $\Gamma \Vdash a : \mathsf{w}_{\rho}\langle \mathrm{B} \rangle$. It follows that $\mathrm{B} <: \mathrm{A}$ and therefore, by Weakening, $\Gamma, \tilde{c} : \tilde{C} \Vdash v : \mathrm{A}$. This means $\Gamma \sqcap v : \mathrm{A}$ is indeed well-defined, and $\Gamma, \tilde{c} : \tilde{C} <: \Gamma \sqcap v : \mathrm{A}$. Applying Weakening again we obtain the required $\Gamma, \tilde{c} : \tilde{C} \Vdash^{\sigma} T'$. □

Note that in this Lemma the requirement $\Gamma \Vdash P$ is essential to ensure that if T receives a value v then that value is compatible with the type environment Γ.

May testing is determined by the *traces*, s, t, in $VAct^*$ which processes can perform. Let ϵ represent the empty trace. The notion of complementary actions lifts element-wise to traces, $\overline{s}$. The names in a trace $\mathsf{n}(s)$ is defined as the union of the names in the individual actions; likewise the bound names in a trace $\mathsf{bn}(s)$ is defined as the union of the bound names in the individual actions.

DEFINITION 5.7 (TRACES). Let $\Gamma \triangleright P \xRightarrow{s}_{\sigma} \Gamma' \triangleright P'$ be the least relation such that:

$$\text{(TR-}\epsilon\text{)}\quad \frac{}{\Gamma \triangleright P \stackrel{\epsilon}{\Longrightarrow}_\sigma \Gamma \triangleright P}$$

$$\text{(TR-}\tau\text{)}\quad \frac{\Gamma \triangleright P \xrightarrow{\tau}_\sigma \Gamma' \triangleright P' \qquad \Gamma \triangleright P' \stackrel{s}{\Longrightarrow}_\sigma \Gamma'' \triangleright P''}{\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma \Gamma'' \triangleright P''}$$

$$\text{(TR-}\alpha\text{)}\quad \frac{\Gamma \triangleright P \xrightarrow{\alpha}_\sigma \Gamma' \triangleright P' \qquad \Gamma' \triangleright P' \stackrel{s}{\Longrightarrow}_\sigma \Gamma'' \triangleright P''}{\Gamma \triangleright P \stackrel{\alpha\cdot s}{\Longrightarrow}_\sigma \Gamma'' \triangleright P''}\ \mathsf{n}(\alpha) \cap \mathsf{bn}(s) = \emptyset$$

We use $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma$ to mean that $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma \Gamma'' \triangleright P''$ for some $\Gamma'' \triangleright P''$. □

We can generalise the function $\mathcal{E}$ from actions to sequences by:

$$\mathcal{E}(\epsilon) = \emptyset \quad \mathcal{E}((\tilde{c}\!:\!\tilde{\mathrm{C}})a?v \cdot s) = \{\tilde{c}\!:\!\tilde{\mathrm{C}}\},\, \mathcal{E}(s) \quad \mathcal{E}((\tilde{c}\!:\!\tilde{\mathrm{C}})a!v \cdot s) = \{\tilde{c}\!:\!\tilde{\mathrm{C}}\},\, \mathcal{E}(s)$$

Note that $\mathcal{E}(s) = \mathcal{E}(\overline{s})$. This notation enables us to generalise the Decomposition Lemma, Lemma 5.6, to traces. The statement assumes a definition of the untyped reductions $P \stackrel{s}{\Longrightarrow} P'$, similar to that in Definition 5.7. We use

PROPOSITION 5.8 (TRACE DECOMPOSITION). *Suppose* $\Gamma \Vdash^{\sigma} T$ *and* $\Gamma \Vdash P$. *Then* $P \,|\, T \xrightarrow{\tau}{}^{*} R$ *implies there exists a trace* s *such that* R *has the form* $(\mathsf{new}\,\mathcal{E}(s))\ (P' \,|\, T')$ *and* $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma \Gamma' \triangleright P'$ *and* $T \stackrel{\overline{s}}{\Longrightarrow} T'$ *and* $\Gamma' \Vdash^{\sigma} T'$. □

Proof. By induction on the length of $P \,|\, T \xrightarrow{\tau}{}^{*} R$, using Lemma 5.6. □

In general the converse to this result is not true; the behaviour of a process P is not determined by the set of sequences s such that $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma$. For example, if Γ allows the value v to be sent and received on channel a at level σ then

$$\mathbf{0} \simeq^{\sigma}_{\Gamma} a?(X)\,\mathbf{0} \ \mid\ a!\langle v\rangle\,.$$

Our language is asynchronous and therefore, as in [18, 7], we need to consider the *asynchronous* actions of processes.

DEFINITION 5.9. (Asynchronous traces) Let $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}{}^{a}_{\sigma} \Gamma' \triangleright Q$ be the least relation which, in addition to the clauses in Definition 5.7, satisfies

$$\frac{\begin{array}{l}(\textsc{c-ain})\\ \Gamma \Vdash a : \mathsf{w}_\delta\langle \mathrm{B}\rangle,\\ \Gamma, \tilde{c} : \tilde{\mathrm{C}} \Vdash v : \mathrm{B},\\ \Gamma, \tilde{c} : \tilde{\mathrm{C}} \triangleright P \mid \delta[\![a!\langle v\rangle]\!] \stackrel{s}{\Longrightarrow}{}^a_\sigma \Gamma \triangleright Q\end{array}}{\Gamma \triangleright P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v.s}{}^a_\sigma \Gamma' \triangleright Q} \quad \begin{array}{l}\delta \preceq \sigma\\ \tilde{c} \notin \mathsf{fn}(P)\end{array}$$

Again we use $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}{}^a_\sigma$ to mean that $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}{}^a_\sigma \Gamma'' \triangleright P''$ for some $\Gamma'' \triangleright P''$. □

The ability to compose asynchronous traces depends on the fact that our language is *asynchronous*. To state the required compositional property we need a structural equivalence on processes. This is least equivalence preserved by the static operators, $\sigma[\![\ ]\!]$, $\mid$ and $(\mathsf{new}\, a)$, generated by the following equations, where for convenience the types of bound variables are omitted.

$$\begin{array}{lrcl}
(\textsc{s-sr}) & P & \equiv_\Gamma & \sigma[\![P]\!] \ \text{ if } \Gamma \Vdash^{\sigma} P\\
(\textsc{s-srsr}) & \sigma[\![\rho[\![P]\!]]\!] & \equiv_\Gamma & (\sigma \sqcap \rho)[\![P]\!]\\
(\textsc{s-srpar}) & \sigma[\![P \mid Q]\!] & \equiv_\Gamma & \sigma[\![P]\!] \mid \sigma[\![Q]\!]\\
(\textsc{s-srnew}) & \sigma[\![(\mathsf{new}\, a)\ P]\!] & \equiv_\Gamma & (\mathsf{new}\, a)\ \sigma[\![P]\!]\\
(\textsc{s-newnew}) & (\mathsf{new}\, a)(\mathsf{new}\, b)\ P & \equiv_\Gamma & (\mathsf{new}\, b)(\mathsf{new}\, a)\ P \ \text{ if } a \neq b\\
(\textsc{s-newpar}) & P \mid (\mathsf{new}\, a)\ Q & \equiv_\Gamma & (\mathsf{new}\, a)\ (P \mid Q) \ \text{ if } a \notin \mathsf{fn}(P)\\
(\textsc{s-comm}) & P \mid Q & \equiv_\Gamma & Q \mid P\\
(\textsc{s-zero}) & P \mid \mathbf{0} & \equiv_\Gamma & P\\
(\textsc{s-iter}) & *P & \equiv_\Gamma & *P \mid P
\end{array}$$

The first three equations allow us to manipulate the typing annotations $\sigma[\![\ ]\!]$, as discussed briefly at the end of Section 3; the remainder are familiar from [21]. We leave to the reader the rather tedious chore of proving that this equivalence is preserved under reductions:

LEMMA 5.10. *If $P \equiv_\Gamma Q$ and $P \xrightarrow{\mu} P'$ then there exists some $Q' \equiv_\Gamma P'$ such that $Q \xrightarrow{\mu} Q'$.* □

LEMMA 5.11 (ASYNCHRONOUS ACTIONS). *If $\Gamma \Vdash^{\sigma} T$ and $T \xrightarrow{(\tilde{c}:\tilde{C}a!v)} T'$ then $T \equiv_\Gamma (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}})\ (\delta[\![a!\langle v\rangle]\!] \mid T')$, for some $\delta \preceq \sigma$.* □

Proof. By induction on the derivation of $T \xrightarrow{(\tilde{c}:\tilde{C}a!v)} T'$. We give two examples.

- $a!\langle v\rangle \xrightarrow{a!v} \mathbf{0}$.

  Since $\Gamma \Vdash^{\sigma} a!\langle v\rangle$ we have $a!\langle v\rangle \equiv_\Gamma \sigma[\![a!\langle v\rangle]\!]$ and the result follows.

- $\rho[\![P]\!] \xrightarrow{(\tilde{c}:\tilde{C}a!v)} \rho[\![P']\!]$ because $P \xrightarrow{(\tilde{c}:\tilde{C}a!v)} P'$.
  $\Gamma \Vdash^{\sigma} \rho[\![P]\!]$ implies $\Gamma \Vdash^{\sigma\sqcap\rho} P$ and so by induction
  $$P \equiv_\Gamma (\mathsf{new}\,\tilde{c}:\tilde{\mathrm{C}})\ (\delta[\![a!\langle v\rangle]\!] \mid P')$$
  for some $\delta \preceq \sigma \sqcap \rho$. Using the rules (S-SRNEW)(S-SRSR) and (S-SRPAR) we can then show $\rho[\![P]\!] \equiv_\Gamma (\mathsf{new}\,\tilde{c}:\tilde{\mathrm{C}})\ (\rho \sqcap \delta[\![a!\langle v\rangle]\!] \mid \rho[\![P']\!])$.

□

PROPOSITION 5.12 (TRACE COMPOSITION). *Suppose* $\Gamma \Vdash^{\sigma} T$. *If* $\Gamma \triangleright P \overset{s}{\Longrightarrow}{}^{a}_{\sigma} \Gamma' \triangleright P'$ *and* $T \overset{\overline{s}}{\Longrightarrow} T'$, *then* $P \mid T \xrightarrow{\tau}{}^{*} (\mathsf{new}\,\mathcal{E}(s))\ (P' \mid T')$. □

Proof. By induction on the derivation $\Gamma \triangleright P \overset{s}{\Longrightarrow}{}^{a}_{\sigma} \Gamma' \triangleright P'$. We examine the most interesting case, when s has the form $\alpha.s'$ and α is the input action $(\tilde{c}:\tilde{\mathrm{C}})a?v$. Further let us assume that the derivation $T \overset{\overline{s}}{\Longrightarrow} T'$ has the form $T \xrightarrow{\overline{\alpha}} T'' \overset{\overline{s'}}{\Longrightarrow} T'$. There are two (interesting) possibilities for the derivation $\Gamma \triangleright P \overset{s}{\Longrightarrow}{}^{a}_{\sigma} \Gamma' \triangleright P'$.

- $\Gamma \triangleright P \xrightarrow{\alpha}_{\sigma} \Gamma' \triangleright P'' \overset{s}{\Longrightarrow}{}^{a}_{\sigma} \Gamma' \triangleright P'$. Using Subject Reduction we can show that $\Gamma' \Vdash^{\sigma} T''$, since Γ' is determined by the action α. So we may apply induction to obtain a reduction $P'' \mid T'' \xrightarrow{\tau}{}^{*} (\mathsf{new}\,\mathcal{E}(s'))\ (P' \mid T')$. We also have, by the rule (L-COM), $P \mid T \xrightarrow{\tau} (\mathsf{new}\,\tilde{c}:\tilde{\mathrm{C}})\ (P'' \mid T'')$. By combining these we may easily obtain a required reduction $P \mid T \xrightarrow{\tau}{}^{*} (\mathsf{new}\,\mathcal{E}(s))\ (P' \mid T')$.
- $\Gamma, \tilde{c}:\tilde{C} \triangleright P \mid \delta[\![a!\langle v\rangle]\!] \overset{s'}{\Longrightarrow}{}^{a}_{\sigma} \Gamma' \triangleright P'$, where $\Gamma \Vdash a : \mathsf{w}_\delta\langle \mathrm{B}\rangle$ and $\Gamma, \tilde{c}:\tilde{\mathrm{C}} \Vdash v : \mathrm{B}$. Again we can apply induction to obtain a derivation $(P \mid \delta[\![a!\langle v\rangle]\!]) \mid T'' \xrightarrow{\tau}{}^{*} (\mathsf{new}\,\mathcal{E}(s'))\ (P' \mid T')$. and therefore
  $$(\mathsf{new}\,\tilde{c}:\tilde{C})\ (P \mid \delta[\![a!\langle v\rangle]\!] \mid T'') \xrightarrow{\tau}{}^{*} (\mathsf{new}\,\mathcal{E}(s))\ (P' \mid T').$$
  However we can we apply the previous Lemma to the derivation $T \xrightarrow{\overline{\alpha}} T''$ to obtain the fact that $T \equiv_\Gamma (\mathsf{new}\,\tilde{c}:\tilde{\mathrm{C}})\ (\delta[\![a!\langle v\rangle]\!] \mid T'')$. Moreover since the names $\tilde{c}$ are new to P we have
  $$P \mid T \equiv_\Gamma (\mathsf{new}\,\tilde{c}:\tilde{C})\ (P \mid \delta[\![a!\langle v\rangle]\!] \mid T'')$$
  and the result follows.

□

These two results immediately give us a sufficient condition for two processes to be semantically equivalent.

DEFINITION 5.13. We write $\Gamma \Vdash^{\sigma} P \simeq_{aseq} Q$ to mean $\Gamma \triangleright P \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$ if and only if $\Gamma \triangleright Q \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$, for every sequence s. □

THEOREM 5.14. *Suppose* $\Gamma \Vdash P,\ Q$. *Then* $\Gamma \Vdash^{\sigma} P \simeq_{aseq} Q$ *implies* $P \simeq^{\sigma}_{\Gamma} Q$. □

Proof. Immediate from the Trace Composition and Decomposition results. □

5.2 Proof of the Main Result

The proof of the non-interference result will now depend on comparing the traces of the processes P and $P \mid H$. First we must show some properties of σ-free processes.

LEMMA 5.15. *If H is σ-free and $H \xrightarrow{\mu} H'$ then H' is also σ-free.* □

Proof. A simple induction on $H \xrightarrow{\mu} H'$. □

We now show that, in appropriate environments, σ-free processes can never perform σ-level write actions. Unfortunately the proof, which is inductive, requires a slight generalisation of the notion of σ-freedom.

DEFINITION 5.16. We say P is σ-free *relative to* δ if $\rho \not\preceq \sigma$ for everyρ in $\mathrm{sl}_\delta(P)$. □

Note that if P is σ-free *relative to* δ then, since $\delta \in \mathrm{sl}_\delta(P)$, we know that $\delta \not\preceq \sigma$. Also P being σ-free *relative to* top means precisely that P is σ-free.

LEMMA 5.17. *Suppose $\Gamma \Vdash^{\delta} P$, where P is σ-free relative to δ. Then $\Gamma \triangleright P \xrightarrow{\alpha}_\rho \Gamma' \triangleright P'$, where α is an output action, implies $\rho \not\preceq \sigma$.* □

Proof. By induction on the derivation of $\Gamma \triangleright P \xrightarrow{\alpha}_\rho \Gamma' \triangleright P'$. We give the two most important cases.

- $\Gamma \triangleright a!\langle v\rangle \xrightarrow{a!v}_\rho \Gamma \triangleright \mathbf{0}$, because $\Gamma \Vdash a : \mathsf{r}_{\rho'}\langle \mathrm{A}\rangle$ for some $\rho' \preceq \rho$. But from $\Gamma \Vdash^{\delta} a!\langle v\rangle$ we have $\Gamma \Vdash a : \mathsf{w}_\delta\langle \mathrm{B}\rangle$ and by the fact that $\Gamma(a)$ must be a well-defined type $\delta \preceq \rho'$. Since $\delta \not\preceq \sigma$ it follows that $\rho \not\preceq \sigma$.
- $\Gamma \triangleright \epsilon[\![Q]\!] \xrightarrow{\alpha}_\rho \Gamma \triangleright \epsilon[\![Q']\!]$ because $\Gamma \triangleright Q \xrightarrow{\alpha}_\rho \Gamma \triangleright Q$. Here we need to apply induction.

  Note that $\mathrm{sl}_\delta(P) = \{\epsilon \sqcap \delta\} \cup \mathrm{sl}_{\epsilon\sqcap\delta}(Q)$ and therefore Q is σ-free relative to $\epsilon \sqcap \delta$. Moreover $\Gamma \Vdash^{\delta} P$ implies $\Gamma \Vdash^{\epsilon\sqcap\delta} Q$ and therefore induction can be applied to obtain the required $\rho \not\preceq \sigma$.

□

The main technical result required for non-interference is given in the following Proposition:

PROPOSITION 5.18. *Suppose $\Gamma \Vdash^{\sigma} P$ and $\Gamma \Vdash^{\mathsf{top}} H$, where H is σ-free. Then $\Gamma \triangleright P \mid H \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$ implies $\Gamma \triangleright P \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$.* □

Proof. The proof is by induction on the derivation of $\Gamma \triangleright P \mid H \stackrel{s}{\Longrightarrow}{}^a_\sigma$. We examine the most interesting cases.

- $\Gamma \triangleright P \mid H \stackrel{\tau}{\longrightarrow}_\sigma \Gamma \triangleright R \stackrel{s}{\Longrightarrow}{}^a_\sigma$.

  The most important case here is when there is communication between P and H. Here $P \stackrel{\alpha}{\longrightarrow} P'$, $H \stackrel{\overline{\alpha}}{\longrightarrow} H'$, R is $(\mathsf{new}\,\tilde{c}:\tilde{C})\ (P' \mid H')$, where $\tilde{c}$ are the bound variables in α. There are two possibilities.

  - Output from P to H; α has the form $(\tilde{c}:\tilde{C})a!v$. Let us examine the trace $\Gamma \triangleright (\mathsf{new}\,\tilde{c}:\tilde{C})\ (P' \mid H') \stackrel{s}{\Longrightarrow}{}^a_\sigma$. Somewhere in s the names in $\tilde{c}$ may be exported. In general we can construct a related trace s_c such that $\Gamma, \tilde{c}:\tilde{C} \triangleright (P' \mid H) \stackrel{s_c}{\Longrightarrow}{}^a_\sigma$, with the property that for any Q, $\Gamma, \tilde{c}:\tilde{C} \triangleright Q \stackrel{s_c}{\Longrightarrow}{}^a_\sigma$ implies $\Gamma \triangleright Q \stackrel{s_c}{\Longrightarrow}{}^a_\sigma$; s_c is obtained from s by omitting any bounds $(c:C)$ found on its output actions.

    Now we may apply induction to $\Gamma, \tilde{c}:\tilde{C} \triangleright (P' \mid H') \stackrel{s_c}{\Longrightarrow}{}^a_\sigma$, since $\Gamma \Vdash^\sigma P'$ by Subject Reduction and $\Gamma, \tilde{c}:\tilde{C} \Vdash^{\mathsf{top}} H'$ by Lemma 5.4. This gives $\Gamma, \tilde{c}:\tilde{C} \triangleright P' \stackrel{s_c}{\Longrightarrow}{}^a_\sigma$.

    Applying Lemma 5.11 we know that P is structurally equivalent to $(\tilde{c}:\tilde{C})(a!\langle v\rangle \mid P')$. Trivially $\Gamma, \tilde{c}:\tilde{C} \triangleright (a!\langle v\rangle \mid P') \stackrel{s_c}{\Longrightarrow}{}^a_\sigma$ from which it follows immediately that $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}{}^a_\sigma$.
  - Output from H to P. We show that this case is not possible as it would involve a write down. Here α would have the form $(\tilde{c}:\tilde{C})a?v$ and applying Subject Reduction to both $\Gamma \Vdash^\sigma P$ and $\Gamma \Vdash^{\mathsf{top}} H$ we would obtain both $\Gamma \Vdash a:\mathsf{r}_\sigma\langle\mathrm{A}\rangle$ and $\Gamma \Vdash a:\mathsf{w}_{\mathsf{top}}\langle\mathrm{B}\rangle$. Since Γ is a well-defined type this would imply $\mathsf{top} \preceq \sigma$, which contradicts the fact that H is σ-free.

- $\Gamma \triangleright P \mid H \stackrel{\alpha}{\longrightarrow}_\sigma \Gamma' \triangleright R \stackrel{s'}{\Longrightarrow}{}^a_\sigma$, where α is an input action $(\tilde{c}:\tilde{C})a?v$.

  Here, again, there are two possibilities depending on which of P, H performs the input move. In the former case a simple argument by induction suffices. If on the other hand it is H, an application of induction gives $\Gamma' \triangleright P \stackrel{s'}{\Longrightarrow}{}^a_\sigma$.

  However from the inference $\Gamma \triangleright H \stackrel{\alpha}{\longrightarrow}_\sigma \Gamma' \triangleright H$ we know that $\Gamma \Vdash a:\mathsf{w}_\delta\langle\mathrm{A}\rangle$, for some $\delta \preceq \sigma$ and some A such that $\Gamma' \Vdash v:\mathrm{A}$. From the result of the application of induction we can deduce $\Gamma' \triangleright (\delta[\![a!\langle v\rangle]\!] \mid P) \stackrel{s'}{\Longrightarrow}{}^a_\sigma$; This is sufficient for us to apply Definition 5.9 to conclude $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}{}^a_\sigma$.
- $\Gamma \triangleright P \mid H \stackrel{\alpha}{\longrightarrow}_\sigma \Gamma' \triangleright \stackrel{s'}{\Longrightarrow}{}^a_\sigma$, where α is an output action $(\tilde{c}:\tilde{C})a!v$.

  Here Lemma 5.17 implies that H can not be responsible for the action; it must be P, and again a simple inductive argument suffices.
- s has the form $\alpha.s'$, where α is an input action $(\tilde{c}:\tilde{C})a?v$, and $\Gamma,\ \tilde{c}:\tilde{C} \triangleright P \mid H \mid \delta[\![a!\langle v\rangle]\!] \stackrel{s'}{\Longrightarrow}{}^a_\sigma$, because $\Gamma \Vdash a:\mathsf{w}_\delta\langle\mathrm{B}\rangle$ and $\Gamma, \tilde{c}:\tilde{\mathrm{C}} \Vdash v:\mathrm{B}$.

  Since $\Gamma,\ \tilde{c}:\tilde{C} \Vdash^\sigma (P \mid a!\langle v\rangle)$ we may apply induction to obtain

$\Gamma,\, \tilde{c}:\tilde{C} \triangleright (P \mid \delta[\![a!\langle v\rangle]\!]) \overset{s'}{\Longrightarrow}{}^{a}_{\sigma}$. Again we may use now Definition 5.9 to obtain the required $\Gamma,\, \tilde{c}:\tilde{C} \triangleright P \overset{s'}{\Longrightarrow}{}^{a}_{\sigma}$.

□

Given this technical result, we can now prove the Non-Interference Theorem.

THEOREM (5.3). *If* $\Gamma \Vdash^{\sigma} P, Q$ *and* $\Gamma \Vdash^{\mathsf{top}} H, K$ *where* H, K *are* σ*-free processes, then:*

$$P \simeq^{\sigma}_{\Gamma} Q \textit{ implies } P \mid H \simeq^{\sigma}_{\Gamma} Q \mid K.$$

□

Proof. To establish the result, as has already been explained, it is sufficient to show that $P \simeq^{\sigma}_{\Gamma} P \mid H$. In fact by Theorem 5.14 it is sufficient to show $\Gamma \triangleright P \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$ implies $\Gamma \triangleright P \mid H \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$, which is immediate, and $\Gamma \triangleright P \mid H \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$ implies $\Gamma \triangleright P \overset{s}{\Longrightarrow}{}^{a}_{\sigma}$; this follows from the previous Proposition. □

Note that the requirement that P, Q be well-typed processes at level σ is necessary for this result to be true. For example consider the process P defined by $h?(x)\, l?y.\, \mathbf{0}$ in an environment Γ in which h, l are high-level and low-level resources respectively. Then $P \simeq^{\mathsf{bot}}_{\Gamma} \mathbf{0}$. However $P \mid H \not\simeq^{\mathsf{bot}}_{\Gamma} H$, where H is the high-level process $h!\langle\rangle$.

6 Conclusions and Related Work

In this paper we have proposed a simple typing system for enforcing a variety of security properties for the *security* π*-calculus*. The types are obtained by adding security levels to the standard input/output types of the π-calculus, [23, 25]. The main novelty is a uniform typing system, a simple extension to that in [23] which can handle two disparate security issues, by a minor variation in the set of types. The first set, called *R-Types*, is designed with resource access control in mind; the security level of a resource (or more formally a capability on a resource) dictates the security clearance required by any process seeking to access that resource. In future work we hope to extend these types for use in distributed systems, [26]. The second set, the more restricted *I-types*, controls the (implicit) flow of information from high to low security levels; this is formalised via a non-interference result for *may* testing equivalence over our *security* π*-calculus.*

There is considerable tension between the expressiveness of the language under investigation, the restrictiveness of the type system and the strength of the non-interference result possible. The π-calculus is very expressive and consequently there are many different ways in which a context may discern a difference in process behaviours. For example, as we have seen, the context can test if a process has the ability (or not) to rendezvous on a specific channel. These types of distinguishing contexts are not available in sequential languages, such as those studied in [30, 6]. Consequently to ensure non-interference type systems for the π-calculus have to be much more restrictive than for sequential or simple multi-threaded languages. Our type system establishes non-interference with respect to a relatively weak equivalence, *may* testing. Stronger results, in the sense of non-interference with respect to more discriminating equivalences such as *must* testing or *observational equivalence*, might be obtained by restricting further the ability to be well-typed. For example we could introduce types which ensure that there is no contention between high-level and low-level processes over read access to channels or types which ensure that when a high-level process reads a value from a low-level channel it immediately restores it. This line of research has been pursued further in [19] where, at the expense of extending considerably the syntax of the π-calculus, they have introduced a much more sophisticated type system, which includes, *linear, receptive* types and adaptations of the *behaviour* types from [31]. They hope to establish non-interference theorems with respect to some notion of bisimulation equivalence (which is much stronger than testing) but the precise details have yet to be published. Nevertheless there is a danger in this approach. As the sophistication of the type system increases not only do type checking and type inference become more complicated, there is also the likelihood that much of the expressive power of the underlying language is lost.

An alternative (and much more established) approach, [30] starts with a much less expressive language (essentially a sequential language of while programs) and by using a relatively weak type system obtains a relatively strong non-interference result - at least as formulated in [6]. But as the expressiveness of the language is increased, [29, 6] the restricting power of the type system must in turn be increased, to rule out more potentially interfering behaviours, and in turn the non-interference results can only be obtained with respect to weaker equivalences. At this point it is fair to say that the relative importance of the parameters

- expressiveness of the language
- expressiveness of the type system
- strength of equivalence used in non-interference

remains to be elucidated. But we believe that the *security π-calculus* is an excellent vehicle in which such questions can be explored.

Methods for controling information flow are a central research issue in computer security [8, 15, 29] and in the Introduction we have indicated a number of different approaches to its formalisation. Non-interference has emerged as a useful concept and is widely used to infer (indirectly) the absence of information flow. In publications such as [27, 10] it has been pointed out that process algebras may be fruitfully used to formalise and investigate this concept; for example in [9] process algebra based methods are suggested for investigating security protocols, essentially using a formalisation of non-interference for CCS.

However in these publications the *non-interference* is always defined behaviourally, as a condition on the possible traces of CCS or CSP processes; useful surveys of trace based non-interference may be found in [10, 28]. Here, we work with the more expressive π-calculus, which allows dynamic process creation and network reconfiguration. Our approach to *non-interference* is also more extensional in that it is expressed in terms of how processes effect their environments, relative to a particular behavioural equivalence. However the proof of our main result, Theorem 5.3, describes may equivalence in terms of (typed) traces; presumably a trace based definition of *non-interference*, similar in style to those in [10, 28] could be extracted from this proof.

More importantly our approach differs from much of the recent process calculus based security research in that we develop purely *static* methods for ensuring security. Processes are shown to be secure not by demonstrating some property of trace sets, using a tool as such as that in [11], but by type-checking. The long term hope is that type systems such as these will be incorporated into *security-aware* programming languages. Thus users working at a given security level would automatically have their applications type checked at that level; moreover the variety of types used, e.g. *R-types* versus *I-types*, could vary according to the particular application.

Types have also been used in this manner in [1] for an extension of the π-calculus called the *spi-calculus*. But there the structure of the types are very straightforward; the type *Secret* representing a secret channel, the type *Public* representing a public one, and *Any* which could be either. However the main interest is in the type rules for the encryption/decryption primitives of the *spi-calculus*. The non-interference result also has a different formulation to ours; it states that the behaviour of well-typed processes is invariant, relative to *may* testing, under certain value-substitutions. Intuitively, it means that the encryption/decryption primitives preserve values of type *Secret* from certain kinds of attackers. It would be interesting to add these primitives to the our *security π-calculus* and to try to adapt the associated type rules to the set of *I-Types*.

Acknowledgements: The research was partially funded by EPSRC grant GR/L93056, and ESPRT Working Group Confer2. The authors would like to thank I. Castellani for a careful reading of a draft version of the paper and the referees for their many suggestions for improvement.

References

[1] Martín Abadi. Secrecy by typing in security protocols. In *Proceedings of TACS'97*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–637. Springer Verlag, 1997.

[2] D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical report MTR-2997, MITRE Corporation, 1975.

[3] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Control flow analysis for the π-calculus. In *Proc. CONCUR'98*, number 1466 in Lecture Notes in Computer Science, pages 84–98. Springer-Verlag, 1998.

[4] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis of processes for no read-up and no write-down. In *Proc. FOSSACS'99*, number 1578 in Lecture Notes in Computer Science, pages 120–134. Springer-Verlag, 1999.

[5] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA-Sophia Antipolis, 1992.

[6] G. Boudol and I. Castellani. Noninterference for concurrent programs. In *ICALP'01*, number 2076 in Lecture Notes in Computer Science, pages 382–395, 2001.

[7] Ilaria Castellani and Matthew Hennessy. Testing theories for asynchronous languages. In V Arvind and R Ramanujam, editors, *18th Conference on Foundations of Software Technology and Theoretical Computer Science (Chennai, India, December 17–19, 1998)*, LNCS 1530. Springer-Verlag, December 1998.

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

[9] Riccardo Focardi, Anna Ghelli, and Roberto Gorrieri. Using non interference for the analysis of security protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[10] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1), 1995.

[11] Riccardo Focardi and Roberto Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23, 1997.

[12] Riccardo Focardi and Roberto Gorrieri. Non interference: Past, present and future. In *Proceedings of DARPA Workshop on Foundations for Secure Mobile Code*, 1997.

[13] 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.

[14] R. Reitmas G. Andrews. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems*, 2(1):56–76, 1980.

[15] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and privacy*, 1992.

[16] 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.

[17] Matthew Hennessy and James Riely. Information flow vs resource access in the asynchronous pi-calculus (extended abstract). In U. Montanari, J. Rolim, and E. Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium*, volume 1853 of *Lecture Notes in Computer Science*, pages 415–427, Geneva, Switzerland, 9–15 July 2000. Springer-Verlag.

[18] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In P. Wegner M. Tokoro, O. Nierstrasz, editor, *Proceedings of the ECOOP '91 Workshop on Object-Based Concurrent Computing*, volume 612 of *LNCS 612*. Springer-Verlag, 1992.

[19] Kohei Honda, Vasco Vasconcelos, and Nobuko Yoshida-Honda. Secure information flow as typed process behaviour. In *Proceedings of European Symposium on Programming (ESOP) 2000*. Springer-Verlag, 2000.

[20] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[21] R. Milner, J. Parrow, and D. Walker. Mobile logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[22] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.

[23] 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.

[24] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.

[25] James Riely and Matthew Hennessy. Resource access control in systems of mobile agents (extended abstract). In *Proceedings of 3rd International Workshop on High-Level Concurrent Languages*, Nice, France, September 1998. Full version to appear in Information and Computation.

[26] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents (extended abstract). In *Conference Record of POPL '99 The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 93–104, 1999.

[27] A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through determinism. In *European Symposium on Research in Computer Security*, volume 875 of *LNCS*, 1994.

[28] P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. In *CSFW 12*. IEEE, 1997.

[29] 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.

[30] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 1996.

[31] Nobuko Yoshida. Graph types for monadic mobile processes. In *FSTTCS*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer-Verlag, 1996.