Information Flow vs. Resource Access in the Asynchronous Pi-Calculus (Extended Abstract)

Matthew Hennessy[1] and James Riely[2]

[1] COGS, University of Sussex, UK
email:matthewh@cogs.sussex.ac.uk,
home page: http://www.cogs.susx.ac.uk/users/matthewh/
[2] DePaul University, US
email:jriley@cs.depaul.edu
home page: http://www.depaul.edu/~jriely/

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.

We 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, for a certain class of processes, 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.

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 [13] while in [27, 15] type systems have been developed for a number of prototypical programming languages. In this paper, we explore the extent to which type systems for ensuring various forms of security can also be developed for the asynchronous π-calculus [5, 17]. We discuss two security issues: resource access control and information control. The former is described in terms of runtime errors, the latter in terms of non-interference [27, 11].

The (asynchronous) π-calculus is a very expressive language for describing distributed systems, [5, 23, 12], 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.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 415-427, 2000

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 [22, 24], 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 $P \stackrel{\Sigma}{\longmapsto} \mathit{err}$, 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 $Q \stackrel{\Gamma}{\not\longmapsto} \mathit{err}$.

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 appropriate.

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:

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

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*, [14,

25, 11, 26], 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, 7]:

- *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 inference 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 [19] or *must testing* equivalence [21]. 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:

$$\text{If } \Gamma \Vdash^{\mathsf{gr}} P, Q \text{ and } \Gamma \Vdash^{\mathsf{top}} H, K \text{ then } P \simeq^{\sigma}_{\Gamma} Q \text{ implies } P \,|\, H \simeq^{\sigma}_{\Gamma} Q \,|\, K \quad (\star\star)$$

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. 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 [6] which in turn depends on a more explicit operational semantics for our language, where actions are paramterised relative to a typing environment. The details may be found in the full version of the paper, [16].

Fig. 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

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 *basic values* BV_σ; we use bv to range over base values. We require that all syntactic sets be disjoint.

The binding constructs $u?(X:\mathrm{A})\,Q$ and $(\mathsf{new}\, a:\mathrm{A})\ Q$ introduce the usual notions of free names and variables, $\mathsf{fn}(P)$ and $\mathsf{fv}(P)$, respectively, and associated notions of substitution; details may be found in the full version. Moreover the typing annotations on the binding constructs, which will be explained in Section 3, are omitted whenever they do not play a role.

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 $\tilde{c}$

Visible actions (all except τ) are ranged over by α, β and we 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*, with $\overline{\alpha}$ denoting the complement of α.

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 [20], to which the reader is referred for more motivation.

Informally a security policy associates with each input/output capability on a channel a security level. To this end, *Pre-capabilities* and *pre-types* are defined

Fig. 2 Labelled Transition Semantics

$$\text{(L-OUT)}\quad \frac{}{a!\langle v\rangle \xrightarrow{a!v} \mathbf{0}} \qquad \text{(L-IN)}\quad \frac{}{a?(X)\,P \xrightarrow{(\tilde{c}:\mathrm{C})a?v} P\{^{v}/_{X}\}}\ \tilde{c}\notin \mathsf{fn}(P), \tilde{c}\in \mathsf{fn}(v)$$

$$\text{(L-OPEN)}\quad \frac{P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P'}{(\mathsf{new}\, b:\mathrm{B})\ P \xrightarrow{(b:\mathrm{B})(\tilde{c}:\mathrm{C})a!v} P'}\ b\neq a,\ b\in \mathsf{fn}(v)$$

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

$$\text{(L-EQ)}\quad \frac{}{\mathsf{if}\ u=u\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P} \qquad \frac{}{\mathsf{if}\ u=w\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q}\ u\neq w$$

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

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

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 $\{\mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle,\ \mathsf{w}_{\mathsf{top}}\langle \mathrm{B}\rangle\}$ 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 $P \xrightarrow{\Sigma} err$. For example, relative to the policy Σ defined above, after one reduction step of the process $\mathsf{top}[\![c!\langle \mathsf{hl}\rangle]\!] \mid \mathsf{bot}[\![c?(x)\, x!\langle v\rangle]\!]$, there is a security error because

Fig. 3 Runtime Errors

$$\text{(E-RD)}\quad \rho[\![a?(X)\,P]\!] \xrightarrow{\Sigma} err \qquad \text{if } \sigma \preceq \rho \text{ implies for all A, } \mathsf{r}_\sigma\langle \mathrm{A}\rangle \notin \Sigma(a)$$

$$\text{(E-WR}_1\text{)}\quad \rho[\![a!\langle v\rangle]\!] \xrightarrow{\Sigma} err \qquad \text{if } \sigma \preceq \rho \text{ implies for all A, } \mathsf{w}_\sigma\langle \mathrm{A}\rangle \notin \Sigma(a)$$

$$\text{(E-WR}_2\text{)}\quad \rho[\![a!\langle v\rangle]\!] \xrightarrow{\Sigma} err \qquad \text{if } bv \in v,\ bv \in \mathbf{B}_\sigma \text{ and } \sigma \not\preceq \rho$$

$$\text{(E-STR)}\quad \frac{P \xrightarrow{\Sigma} err}{P \mid Q \xrightarrow{\Sigma} err} \qquad \frac{P \xrightarrow{\Sigma} err}{\rho[\![P]\!] \xrightarrow{\Sigma} err} \qquad \frac{P \equiv Q,\ P \xrightarrow{\Sigma} err}{Q \xrightarrow{\Sigma} err}$$

$$\frac{P \xrightarrow{\Sigma, a:\mathrm{A}} err}{(\mathsf{new}\, n:\mathrm{A})\ P \xrightarrow{\Sigma} err}$$

$\mathsf{bot}[\![\mathsf{hl}!\langle v\rangle]\!] \xrightarrow{\Sigma} err$. A low security process has read 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.

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 $RType_\sigma$ 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 1. *Let* $<\!:$ *be the least preorder on pre-capabilities and pre-types such that:*

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

For each ρ, *let* RType_ρ *be the least set that satisfies:*

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

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

$$\text{(RT-BASE)}\quad \frac{}{\mathbf{B}_\sigma \in \mathrm{RType}_\rho}\ \sigma \preceq \rho$$

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

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

Fig. 4 Typing Rules

$$\frac{\Gamma(u) <: \mathrm{A}}{\Gamma \vdash u : \mathrm{A}} \text{(T-ID)} \qquad \frac{bv \in \mathbf{B}_\sigma}{\Gamma \vdash bv : \mathbf{B}_\sigma} \text{(T-BASE)} \qquad \frac{\Gamma \vdash v_i : A_i \quad (\forall i)}{\Gamma \vdash (v_1, \ldots, v_k) : (\mathrm{A}_1, \ldots, \mathrm{A}_k)} \text{(T-TUP)}$$

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

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

$$\frac{\Gamma \Vdash^{\sigma \sqcap \rho} P}{\Gamma \Vdash^\sigma \rho[\![P]\!]} \text{(T-SR)} \qquad \frac{\Gamma, a : \mathrm{A} \Vdash^\sigma P}{\Gamma \Vdash^\sigma (\mathsf{new}\, a : \mathrm{A})\, P} \text{(T-NEW)} \qquad \frac{\Gamma \Vdash^\sigma P,\, Q}{\Gamma \Vdash^\sigma P \mid Q,\ *P,\ \mathbf{0}} \text{(T-STR)}$$

Let RType *be the union of the kinds* RType_ρ *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 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 $RType_{\mathsf{bot}}$. The compatibility requirement between read and write capabilities in a type (RT-WRRD), in addition to the typing implications discussed in [24], 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. 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 1. *For every* ρ, RType_ρ *is a preorder with respect to* $<:$, *with both a partial meet operation* $\sqcap$ *and a partial join* $\sqcup$. □

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. 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.

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 [24, 22] for the standard IO typing of the π-calculus. Indeed 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.

Theorem 1 (Subject Reduction). *Suppose $\Gamma \models^{\sigma} P$. Then*

- *$P \xrightarrow{\tau} Q$ implies $\Gamma \models^{\sigma} Q$*
- *$P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v} Q$ 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 $\Gamma \sqcap v : \mathrm{A} \models^{\sigma} Q$.*
- *$P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$ 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 $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \models^{\sigma} Q$.*

□

We can now state the first main result:

Theorem 2 (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 $Q \overset{\Gamma}{\not\longmapsto}$ err*

□

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 $\Gamma \models^{\sigma} P$.

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. 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 considered in the Introduction, $(\star)$. 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 2. *For each ρ, let* IType_ρ, *be the least set that satisfies the rules in Definition 1, with* (RT-WRRD) *replaced by:*

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

Let IType *be the union of* IType_ρ *over all* ρ. *We write* $\Gamma \Vdash^{\mathsf{g}} P$ *if* $\Gamma \vdash^{\mathsf{g}} P$ *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 3. *Define the security levels of a term below* ρ, $\mathrm{sl}_\rho(P)$, *as follows:*

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

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

Non-interference, as discussed in the Introduction, $(\star\star)$, depends on a formulation of a behavioural equivalence, 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

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

It is very easy to check that $\Gamma \Vdash P, H$ and that H is bot-free. Note that in the term $P \mid H$ 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*, [19], or *must testing*, [21], 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 $P \mid \mathbf{0}$. Using the terminology of [21], $P \mid \mathbf{0}$ *guarantees* the test $\mathsf{bot}[\![a?(x)\,\omega!\langle\rangle]\!]$ whereas $P \mid H$ does not.

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 $\Gamma \Vdash^{\mathsf{g}} \omega!\langle\rangle$ for all Γ. 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. We write $T\Downarrow$ if $T \xrightarrow{\tau}{}^{*} T'$, where T' has the form $(\mathsf{new}\,\tilde{c})\ (\omega!\langle\rangle \mid T'')$ for some T'' and $\tilde{c}$; that is T can eventually report success.

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 4. *We write $P \simeq^{\sigma}_{\Gamma} Q$ if for every test T such that $\Gamma \Vdash^{\mathsf{g}} T$:*

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

□

We can now state the main result of the paper.

Theorem 3 (Non-Interference). *If $\Gamma \Vdash^{\mathsf{g}} P, Q$ and $\Gamma \Vdash^{\mathsf{top}} H, K$ where H and K are σ-free processes, then $P \simeq^{\sigma}_{\Gamma} Q$ implies $P \mid H \simeq^{\sigma}_{\Gamma} Q \mid K$.* □

The proof of the theorem relies on a constructing sufficient condition to guarantee that two processes are may equivalent. This condition involves the *asynchronous* sequences of actions which processes can perform in the type environment Γ. The details may be found in the full version of the paper, [16], which also contains the subsequent proof of the non-interference result.

Finally let us remark that if we allowed *synchronous* tests then this result would no longer hold. For an appropriate Γ would have:

$$P \mid \mathbf{0} \simeq^{\sigma}_{\Gamma} P \mid H$$

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.

5 Conclusions and Related Work

Methods for controling information flow are a central research issue in computer security [7, 14, 27] 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 [25, 9] it has been pointed out that process algebras may be fruitfully used to formalise and investigate this concept; for example in [8] 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 [9, 26]. 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 3, describes may equivalence in terms of (typed) traces; presumably a trace based definition of *non-interference*, similar in style to those in [9, 26] 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 [10], but by type-checking. 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*.

An extension of the π-calculus is also considered in [18], where a sophisticated type system is used to control information flow. The judgements in their system take the form

$$\Gamma \vdash_s P \triangleright A$$

where s is a security level, P is a process term, A is a poset of so-called *action nodes* and Γ is a type environment. Their environments are quite similar to ours, essentially associating with channels a version of input/output types annotated with, among other things, security levels. However their intuition, and much of the technical development, is quite different from ours. In summary it appears that our type system addresses information flow within the core π-calculus while the more sophisticated one of [18] controls the flow allowed via the extra syntactic

constructs of their language. However a more thorough comparison between the two systems deserves to be made.

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.

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. 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.
7. D. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20:504–513, 1977.
8. 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.
9. Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1), 1995.
10. 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.
11. Riccardo Focardi and Roberto Gorrieri. Non interference: Past, present and future. In *Proceedings of DARPA Workshop on Foundations for Secure Mobile Code*, 1997.
12. 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.
13. R. Reitmas G. Andrews. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems*, 2(1):56–76, 1980.
14. J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and privacy*, 1992.
15. 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.

16. Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. Technical report 2000:03, University of Sussex, 2000. Available from `http://www.cogs.susx.ac.uk/`.
17. 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.
18. 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.
19. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
20. R. Milner, J. Parrow, and D. Walker. Mobile logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.
21. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.
22. 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.
23. 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.
24. 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 available as Computer Science Technical Report 2/98, University of Sussex, 1997. Available from `http://www.cogs.susx.ac.uk/`.
25. 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.
26. P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. In *CSFW 12*. IEEE, 1997.
27. 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.