Electronic Notes in Theoretical Computer Science 83 (2004)
URL: http://www.elsevier.nl/locate/entcs/volume83.html 18 pages

The Security Picalculus and Non-interference (Extended Abstract)

Matthew Hennessy [1,2]

University of Sussex,UK

Abstract

The security π-calculus is a typed version of the asynchronous π-calculus in which the types, in addition to constraining the input/output behaviour of processes, have *security levels* associated with them. This enables us to introduce a range of typing disciplines which allow input or output behaviour, or both, to be bounded above or below by a given security level.

We define typed versions of *may* and *must* equivalences for the security π-calculus, where the tests are parameterised relative to a security level. We provide alternative characterisations of these equivalences in terms of *actions in context*; these describe the actions a process may perform, assuming the observer is constrained by a given typing environment.

Using these alternative characterisations we prove non-interference results with respect to *may* and *must* testing. These show that information flow between security levels can be controlled using our typing systems.

Keywords: Distributed Systems, picalculus, security types, non-interference, testing equivalences.

1 Introduction

The asynchronous π-calculus, [2,13], is a simple formalism for describing distributed processes. It presupposes a set of channel names through which processes communicate. Thus $\mathsf{a}?(X)\,P$ is a process which inputs some value v on the channel a, and executes the body P in which X has been substituted by the value v, while output on the same channel is denoted by $a!\langle v\rangle$. These two primitives, together with operators for parallelism, $|$, repetition, $*$, and channel scoping, $(\mathsf{new}\, n)$, make the π-calculus a very powerful language. For

[1] This research has been partially funded by the EU Global Computing projects *Mikado* and *Myths*
[2] Email: matthewh@cogs.sussex.ac.uk

example the term P,

$$*\,\mathsf{req}?(x,y)\ \ (\mathsf{new}\,r)\ \ \mathsf{s}!\langle x,r\rangle \mid r?(z)\,y!\langle z\rangle$$

describes a process which repeatedly receives a request on the channel req, consisting of a value, bound to x, and a return channel, bound to y. This value is in turn sent along the channel s, presumably serviced by some independent server, together with a private return channel r, generated specifically for this purpose. A response is awaited from the service, on the reply channel r, which is then forwarded on the original return channel y.

Numerous typing systems have been developed for this language, [17,23,24]. Most are based on judgements of the form $\Gamma \vdash P$ indicating that the process P is well-typed with respect to the channel environment Γ, which associates capabilities with the free channel names of P. Usually these capabilities are some elaboration of

read capabilities $\mathsf{r}\langle \mathrm{T}\rangle$**:** the ability to read values at type T from a channel
write capabilities $\mathsf{w}\langle \mathrm{T}\rangle$**:** the ability to write values at type T to a channel

For example let A denote the tuple type $(\mathbf{int}, \mathsf{w}\langle\mathbf{int}\rangle)$; a value of this type will consist of a pair, the first element of which is an integer, and the second a channel on which integers may be written. If Γ associates the type $\mathsf{r}\langle \mathrm{A}\rangle$ with the channel req and the type $\mathsf{w}\langle \mathrm{A}\rangle$ with s, we would expect the above term, P, to be well-typed with respect to Γ. However for this to be true the local channel r needs to be generated with the write capability $\mathsf{w}\langle\mathbf{int}\rangle$, to be sent along the channel s, and the read capability $\mathsf{r}\langle\mathbf{int}\rangle$, which is used by the process itself. Thus if we were to annotate all bound names and variables with their required types we would obtain the annotated term

$$*\,\mathsf{req}?(x,y):\mathrm{A}\ \ (\mathsf{new}\,r:\mathrm{R})\ \ \mathsf{s}!\langle x,r\rangle \mid r?(z)\,y!(z)$$

where R is the type $\{\mathsf{w}\langle\mathbf{int}\rangle, \mathsf{r}\langle\mathbf{int}\rangle\}$, more usually written as $\mathsf{rw}\langle\mathbf{int}\rangle$. This term is well-typed with respect to the above mentioned environment Γ.

Intuitively the use of types constrains the behaviour of processes, ensuring no misuse of channels. By defining sophisticated forms of types process behaviour can be more or less constrained, while at the same time the advantages of well-typing can be preserved. For example a form of polymorphism is investigated in [17], while in [11] security levels are associated with capabilities, to obtain so-called *security types*. Suppose we have two security levels, high, denoted by top, and low, denoted by bot. Then we would have capabilities of the form $\mathsf{r}_{\mathsf{top}}\langle \mathrm{T}\rangle$, $\mathsf{r}_{\mathsf{bot}}\langle \mathrm{T}\rangle$, $\mathsf{w}_{\mathsf{top}}\langle \mathrm{T}\rangle$, $\mathsf{w}_{\mathsf{bot}}\langle \mathrm{T}\rangle$, where T in turn is a security type. By varying the precise definition of a security type we can either implement *resource access control* methodologies, or ensure forms of *non-interference*, [1,7,6]. In this paper we will be concerned with the latter, using a mild variation of the I-types of [11]; essentially types are sets of read/write capabilities, where in addition each capability is annotated by a security level taken from

some complete lattice . We will refer to the asynchronous π-calculus, augmented with these types, as the *security π-calculus*.

The statement of non-interference results requires some definition of *process behaviour*; intuitively a system is interference-free if its low level behaviour is independent of changes to high-level behaviour. The main topic of this paper is an investigation of the notion of behaviour of process, relative to a security level, for the security π-calculus.

Process behaviour is relative to some typing environment Γ and therefore we wish to develop a relation of the form $\Gamma \rhd^{\sigma} P \simeq Q$ meaning, intuitively, that in the typing environment Γ, both P and Q exhibit the same σ-level behaviour. By this we mean that a σ-level observer will be unable to discern a difference between P and Q. For example low-level observers will be unable to see any high-level actions performed by P, Q. But more importantly we assume that these observers are constrained by the typing environment Γ and therefore actions disallowed by this environment will also be invisible to observers.

In this paper we investigate *may* and *must* testing equivalences, [15,8], for the security π-calculus. In particular we give an alternative characterisation of these behavioural equivalences which, as might be expected from [15,8], are based on the (asynchronous) sequences of actions that a process can perform. But here these sequences are relative to both a security level and a typing environment.

These actions, which we call *actions in context*, take the form

$$\Gamma \rhd P \xrightarrow{\mu}_{\sigma} \Gamma' \rhd P'$$

indicating that the process P can perform the action μ to interact with some σ-level observer constrained by the environment Γ; this action affects the process P but may also change the the typing environment of the observing process, from Γ to Γ'.

The remainder of this extended abstract is organised as follows. In Section 2 we formally define the types we use, together with the syntax of the security π-calculus. This is followed with a range of typing systems. In the most straightforward we have the judgements $\Gamma \vdash P$ where Γ is a type environment, associating types to channel names and variables. This means that relative to Γ, P uses its channels correctly as input/output devices, ignoring their security annotations. We also have judgements of the form $\Gamma \vdash_{\sigma} P$ which indicates that in addition P uses channels with security level *at most* σ. Similarly we have a typing relation $\Gamma \vdash^{\sigma} P$ meaning that P uses channels with *at least* security level σ. Indeed we can go further, designing relations such as $\Gamma \vdash_{r\sigma} P$ or $\Gamma \vdash^{w\sigma} P$ where the *read capabilities* or the *write capabilities* of processes are independently constrained. For all of these typing relations Subject Reduction is easily established.

Section 3 is the heart of the paper. First the behavioural preorders and equivalences are defined, by adapting the standard framework, [15,8], to the

security π-calculus. We obtain the relations

$$\Gamma \rhd_\sigma P \simeq_{may} Q \qquad \text{and} \qquad \Gamma \rhd_\sigma P \simeq_{must} Q$$

indicating that P and Q can not be distinguished, relative to *may/must* experiments respectively, by any testing process T such that $\Gamma \vdash_\sigma T$; that is any test running at security level *at most* σ, relative to the type environment. This is followed by an exposition of the actions in context, sequences of which are used the alternative characterisation of $\simeq_{may}$; we also state a much more complicated characterisation of $\simeq_{must}$.

One benefit of having behavioural equivalences relativised to security levels is that non-interference results can be stated succinctly. Section 4 contains two such statements. The first gives conditions ensure that

$$\Gamma \rhd_\sigma P \simeq_{may} Q \text{ implies } \Gamma \rhd_\sigma P \mid H \simeq_{may} Q \mid K.$$

It turns out to be sufficient to require that the *read capabilities* of P and Q be bounded above by σ, that is $\Gamma \vdash_{r\sigma} P, Q$, and that the *write capabilities* of H and K be bounded below by some $\delta \not\preceq \sigma$, that is $\Gamma \vdash^{w\delta} H, K$.

This is quite a general non-interference result. For example in the case where Q is P and K is the empty process $\mathbf{0}$ we obtain

$$\Gamma \rhd_\sigma P \simeq_{may} P \mid H$$

indicating that, under the conditions of the theorem, the process H can not interfere with the behaviour of P.

This result is not true for the *must* equivalence. As explained in Section 4, this is because our types allow contention between processes running at different security levels over read access to channels. However by restricting the type system, to confine *read capabilities* to a single security level, we show that a similar result holds for $\simeq_{must}$.

In this extended abstract all proofs are omitted. They may be found in the full version of the paper, [9].

2 The Language

We presuppose a complete lattice $\langle SL, \preceq, \sqcap, \sqcup, \mathsf{top}, \mathsf{bot}\rangle$ of security annotations, ranged over by $\sigma, \rho, \ldots$. For each σ we assume a set of *basic types* at that level, of the form $\mathbf{B}_\sigma$. If the security annotation is omitted, as in **int**, then we assume it has security level bot; as we shall see values of these types are available to all processes. Also, as explained in the Introduction, a σ-level channel type, for channels accessible to processes with security clearance at level σ, consists of a set of σ-level capabilities, i.e. a subset of Cap_σ. These may either be a read capability, of the form $\mathsf{r}_\rho\langle \mathrm{T}\rangle$, for some appropriate ρ and T, or a write capability, of the form $\mathsf{w}_\sigma\langle \mathrm{T}\rangle$. These capabilities are constrained by

Fig. 1 Types

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

$$\frac{\mathrm{A} \in \mathit{Type}_\sigma}{\mathsf{w}_\sigma\langle \mathrm{A}\rangle \in \mathit{Cap}_\sigma}\ (\text{RT-WR}) \qquad \frac{\mathrm{A} \in \mathit{Type}_\rho}{\mathsf{r}_\rho\langle \mathrm{A}\rangle \in \mathit{Cap}_\sigma}\ \sigma \preceq \rho\ (\text{RT-RD})$$

$$\frac{S \subseteq_{fin} \mathit{Cap}_\sigma}{S \in \mathit{Type}_\sigma}\ S \text{ consistent}\ (\text{RT-WRRD}) \qquad \frac{\mathrm{A}_i \in \mathit{Type}_\sigma \quad (\forall i)}{(\mathrm{A}_1, \ldots, \mathrm{A}_k) \in \mathit{Type}_\sigma}\ (\text{RT-TUP})$$

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

The set of capabilities Cap is *consistent* if

- $\mathsf{w}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{w}_\rho\langle \mathrm{B}\rangle \in$ Cap implies $\sigma = \rho$ and A is B
- $\mathsf{r}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{r}_\sigma\langle \mathrm{B}\rangle \in$ Cap implies A is B
- $\mathsf{w}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{r}_\rho\langle \mathrm{B}\rangle \in$ Cap implies A $<:$ B.

consistency requirements. For example since values with the capability $\mathsf{w}_\sigma\langle \mathrm{T}\rangle$ are written to by σ-level processes we require that T in turn be a σ-level type.

Types, i.e. sets of capabilities, are also constrained. For simplicity in a given type we only allow at most one write capability, and for each level σ at most one read capability at that level. More importantly we ensure that, relative to security levels, only *write-ups*, [7,1], are allowed by requiring that if $\mathsf{w}_\rho\langle \mathrm{T}\rangle$ and $\mathsf{r}_{\rho'}\langle \mathrm{S}\rangle$ are in a type then $\rho \preceq \rho'$; the additional constraint that T be a sub-type of S is well-known [17,12]. The formal definitions of types and capabilities, together with their *subtyping* relation are given in Figure 1. These types correspond very closely to the *I-types* of [11]; the rule (RT-RD) ensures that only *write-ups* are allowed, from low-level processes to high-level processes. But we allow multiple read capabilities, which will enable us to be more flexible with respect to allowing/disallowing reading from a channel at different security levels. However subtyping is more restrictive; unlike [11] they can only be sub-typed at the same security level; $\mathsf{r}_\sigma\langle \mathrm{A}\rangle <: \mathsf{r}_\rho\langle \mathrm{B}\rangle$ only if $\sigma = \rho$. Nevertheless this is compensated for in the existence of multiple read capabilities.

Fig. 2 Syntax

P, Q ::=	*Terms*
$u!\langle v\rangle$	Output
$u?(X:\mathrm{A})\,P$	Input
$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Matching
$(\mathsf{new}\, a:\mathrm{A})\ P$	Name creation
$P \mid Q$	Composition
$*P$	Replication
$\mathbf{0}$	Termination

X, Y ::=	*Patterns*
x	Variable
$(X_1, \ldots, X_k)$	Tuple

u, v, w ::=	*Values*
bv	Base Value
a	Name
x	Variable
$(u_1, \ldots, u_k)$	Tuple

Example 2.1

- The set $\{\mathsf{w}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$ is a bot-level channel type, an element of $Type_{\mathsf{bot}}$; that is channels of this type may be transmitted on bot-level channels. In turn these channels may be written to by a bot-level process or read by either a bot-level or a top-level process.
- The type $\{\mathsf{w}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$ restricts reading from the channel to top-level processes, although bot-level ones can write to it.
- The set $\{\mathsf{w}_{\mathsf{top}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$ is not a valid type as it contains a read capability at a lower level than its write capability.
- The set $\{\mathsf{w}_{\mathsf{top}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$ is a top-level type but not a bot-level one; that is, it is in $Type_{\mathsf{top}}$ but not in $Type_{\mathsf{bot}}$.

Proposition 2.2 *For every σ,* Type_σ *is a preorder with respect to* $<:$*, with both a partial meet operation $\sqcap$ and a partial join $\sqcup$.*

Multiple read capabilities in a type, such as $\{\mathsf{w}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$, allows processes at different security levels to read from the same channel. We can eliminate such contention by using a restricted set of types.

Definition 2.3 [Single-level Types] Let *SlType* be the least set of types obtained by changing the condition on read capabilities in the definition Figure 1 to read:

$\mathsf{r}_\rho\langle\mathrm{A}\rangle, \mathsf{r}_\sigma\langle\mathrm{B}\rangle \in \mathrm{Cap}$ implies $\rho = \sigma$ and A is B.

Note that these types still allow communication from low-level processes to high-level processes. We leave the reader to check that these types, ordered by $<:$ also has both partial meet and join operations.

The syntax of the (asynchronous) π*-calculus*, given in Figure 2, uses a predefined set of *names*, ranged over by $a, b, \ldots$ and a set of *variables*, ranged over by x, y, z. *Identifiers* are either variables or names. We also assume a set

Fig. 3 Typing Rules

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

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

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

$$\frac{\Gamma, a : \mathrm{A} \vdash P}{\Gamma \vdash (\mathsf{new}\, a : \mathrm{A})\ P} \text{(T-NEW)} \qquad \frac{\Gamma \vdash P, Q}{\Gamma \vdash P \mid Q, *P, \mathbf{0}} \text{(T-STR)}$$

of *basic values*, ranged over by bv, each of which belong to a given basic type. We assume the standard notions of free names and variables, $\mathsf{fn}(P)$ and $\mathsf{fv}(P)$, respectively, and associated notions of substitution and α-equivalence, $\equiv_\alpha$, are defined as usual. Moreover the typing annotations on the binding constructs are omitted whenever they do not play a role, as will most occurrences of the empty process $\mathbf{0}$. We also assume the standard reduction semantics for *closed* terms, usually refereed to as *processes*. This is expressed in terms of a binary relation over processes, defined inductively via judgements of the form

$$P \xrightarrow{\tau} Q$$

See the full version [9] for details, or indeed any standard reference to the *picalculus* such as [21].

A *type environment* is a finite mapping from identifiers to types. We adopt some standard notation for describing them. For example, for any identifier u 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, $\Gamma(u) <: \Delta(u)$. 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 arbitrary values v; this is only well-defined when v has the same structure as the type A.

The first typing system is given in Figure 3, where the judgements take the form

$$\Gamma \vdash P$$

Intuitively this means that the process P uses all channels as input/output devices in accordance with their types, as given in Γ. It is the standard typing system for the π-calculus, [17], adapted to our types; note that the security levels on the capabilities do not play any role.

Fig. 4 Security Typing Rules

$$\frac{\Gamma, X:\mathrm{A} \vdash_\sigma P \quad \Gamma \vdash u:\mathsf{r}_\delta\langle \mathrm{A}\rangle}{\Gamma \vdash_\sigma u?(X:\mathrm{A})\,P}\ \delta \preceq \sigma \quad (\text{LT-IN})$$

$$\frac{\Gamma \vdash v:\mathrm{A} \quad \Gamma \vdash u:\mathsf{w}_\delta\langle \mathrm{A}\rangle}{\Gamma \vdash_\sigma u!\langle v\rangle}\ \delta \preceq \sigma \quad (\text{LT-OUT})$$

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

$$\frac{\Gamma, a:\mathrm{A} \vdash_\sigma P}{\Gamma \vdash_\sigma (\mathsf{new}\, a:\mathrm{A})\ P} \quad (\text{LT-NEW})$$

$$\frac{\Gamma \vdash_\sigma P,\, Q}{\Gamma \vdash_\sigma P \mid Q,\, *P,\, \mathbf{0}} \quad (\text{LT-STR})$$

We can also design a type inference system which not only ensures that channels are used according to their types but also controls the security levels of the channels used. One such system is given in Figure 4, where the judgements now take the form

$$\Gamma \vdash_\sigma P$$

This indicates that not only is P well-typed as before but in addition it uses channels with security level *at most* σ. (This corresponds to the typing system used in [11].) The only difference is in the input/output rules, where the security level of the channels used are checked. For example $\Gamma \vdash_\sigma a!\langle v\rangle$ only if in Γ the channel a can be assigned a security level $\delta \preceq \sigma$, in addition to having the appropriate output capability in Γ.

We can also design a typing system

$$\Gamma \Vdash^\sigma P$$

which which ensures that P uses channels with security level *at least* σ. The only change is to demand in the input/output rules that $\sigma \preceq \delta$:

$$\frac{\Gamma, X:\mathrm{A} \Vdash^\sigma P \quad \Gamma \vdash u:\mathsf{r}_\delta\langle \mathrm{A}\rangle}{\Gamma \Vdash^\sigma u?(X:\mathrm{A})\,P}\ \sigma \preceq \delta \quad (\text{HL-IN})$$

$$\frac{\Gamma \vdash u:\mathsf{w}_\sigma\langle \mathrm{A}\rangle \quad \Gamma \vdash v:\mathrm{A}}{\Gamma \Vdash^\sigma u!\langle v\rangle}\ \sigma \preceq \delta \quad (\text{HL-OUT})$$

We can provide further mix and matches. For example the type system

$$\Gamma \vdash_{r\sigma} P$$

ensures that all channels from which values are *read* have a read capability of *at most* σ; the security level of the output channels is unexamined. This system is obtained by using the rules in the original Figure 3 but with the rule (T-IN) replaced with (LT-IN); the output rule is left unchanged. In a similar manner we can define relations $\Gamma \vdash_{w\sigma} P$, $\Gamma \Vdash^{r\sigma} P$ and $\Gamma \Vdash^{w\sigma} P$.

Theorem 2.4 (Subject Reduction) *Let $\Vdash$ denote any of the relations $\vdash$, $\vdash_\sigma$, $\vdash_{r\sigma}$, $\vdash^{r\sigma}$, $\vdash_{w\sigma}$, $\vdash^{w\sigma}$, and suppose $\Delta \Vdash P$. Then $P \xrightarrow{\tau} Q$ implies $\Delta \Vdash Q$.*

3 Behavioural Theories

A *test* or *observer* is a process with an occurrence of a new reserved resource name ω, used to report success. We let T to range over tests, with the typing rule $\Gamma \vdash_\sigma \omega!\langle\rangle$ 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

$$P \textbf{ may } T$$

if $T \mid P \xrightarrow{\tau}{}^* R$ for some R such that R can *report success*, i.e. $R \xrightarrow{\omega!\langle\rangle}$. The stronger relation

$$P \textbf{ must } T$$

holds when in every computation

$$T \mid P \xrightarrow{\tau} R_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} R_n \xrightarrow{\tau} \ldots$$

there is some $R_k,\ k \geq 0$, which can report success.

We can obtain a testing based behavioural preorder between processes by demanding that they react in a similar manner to a given class of tests. Here we choose the class of tests which are well-typed and use channels from *at most* a given security level σ; that is we require that processes react in the same manner to all tests T such that $\Gamma \vdash_\sigma T$.

Definition 3.1 [Testing Preorders] We write $\Gamma \rhd_\sigma P \sqsubseteq_{may} Q$ if for every test T such that $\Gamma \vdash_\sigma T$, P **may** T implies Q **may** T.
Similarly $\Gamma \rhd_\sigma P \sqsubseteq_{must} Q$ means that for every such T, P **must** T implies Q **must** T.
We use $\simeq_{may}$ and $\simeq_{must}$ denote the related equivalence relations.

So for example setting σ to be bot, $\Gamma \rhd_{\mathsf{bot}} P \simeq_{may} Q$ means that in the type environment Γ, P and Q are indistinguishable by low-level observers, from a may testing point of view.

The aim of this section is to outline alternative characterisations of these equivalences. It is well-known, [15,8], that testing equivalences are closely related to the ability of processes to perform sequences of actions. We have explained in the Introduction that here we need to relativise these sequences to security levels and a typing environments for the observers.

The rules for the Context LTS are given in Figure 5. The judgements take the form

$$\Gamma \rhd P \xrightarrow{\mu}_\sigma \Gamma' \rhd P' \tag{1}$$

This judgement should be understood as expressing the fact that:

Fig. 5 Context LTS

(C-OUT)
$$\frac{\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)}{\Gamma \rhd a!\langle v\rangle \xrightarrow{a!v}_\sigma \Gamma \sqcap v:\mathrm{A} \rhd \mathbf{0}} \ \delta \preceq \sigma$$

(C-IN)
$$\frac{\mathsf{w}_\delta\langle \mathrm{A}\rangle \in \Gamma \quad \Gamma \vdash v:\mathrm{A}}{\Gamma \rhd a?(X:\mathrm{B})\, P \xrightarrow{a?v}_\sigma P\{\!|{}^{v}\!/_{X}|\!\}} \ \delta \preceq \sigma$$

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

(C-WEAK)
$$\frac{\Gamma, b:\mathrm{B} \rhd P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v}_\sigma \Gamma' \rhd P'}{\Gamma \rhd P \xrightarrow{(b:\mathrm{B}\tilde{c}:\tilde{\mathrm{C}})a?v}_\sigma \Gamma' \rhd P'} \ b \neq a, \tilde{c}$$

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

(C-CTXT)
$$\frac{\Gamma \rhd P \xrightarrow{\mu}_\sigma \Gamma' \rhd P'}{\Gamma \rhd *P \quad \xrightarrow{\mu}_\sigma \Gamma' \rhd *P \mid P'}$$

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

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

The process P, when run concurrently with any observing process T such that $\Gamma \vdash_\sigma T$, can perform the action μ. This will transform P into P' and may also transform the type environment of the observing process to Γ'.

These actions can take three forms:

internal move: $\Gamma \rhd P \xrightarrow{\tau}_\sigma \Gamma \rhd P'$ This corresponds to an internal move by P, which does not depend on its environment. These moves are completely determined by the reduction semantics.

input move: $\Gamma \rhd P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v}_\sigma \Gamma' \rhd P'$ Here the observing process sends a value v, possibly constructed using the new values $\tilde{c}$ at type $\tilde{\mathrm{C}}$, to P; the type environment of the observing process will be augmented with these new values. An appropriate write capability on a is required of the observing

Fig. 6 (asynchronous) Traces in Context

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

$$\frac{\Gamma \rhd P \xrightarrow{\alpha}_\sigma \Gamma' \rhd P' \quad \Gamma' \rhd P' \stackrel{s}{\Longrightarrow}_\sigma \Gamma'' \rhd P''}{\Gamma \rhd P \stackrel{\alpha \cdot s}{\Longrightarrow}_\sigma \Gamma'' \rhd P''} \ (\text{TR-}\alpha)$$

$$\frac{\mathsf{w}_\delta\langle \mathrm{A} \rangle \in \Gamma \quad \Gamma \vdash v : \mathrm{A}}{\Gamma \rhd P \stackrel{a?v}{\Longrightarrow}_\sigma \Gamma \rhd P \mid a!\langle v \rangle} \ \delta \preceq \sigma \ (\text{TR-ASYNC}-\text{IN})$$

$$\frac{\Gamma, b : \mathrm{B} \rhd P \stackrel{(\tilde{c} : \tilde{\mathrm{C}})a?v}{\Longrightarrow}_\sigma \Gamma' \rhd P'}{\Gamma \rhd P \stackrel{(b : \mathrm{B}\tilde{c} : \tilde{\mathrm{C}})a?v}{\Longrightarrow}_\sigma \Gamma' \rhd P'} \ b \neq a, \tilde{c} \ (\text{TR-ASYNC}-\text{WEAK})$$

process for the action to take place; see the rule (C-IN).

output move: $\Gamma \rhd P \stackrel{(\tilde{c})a!v}{\longrightarrow}_\sigma \Gamma' \rhd P'$ Here P sends a value v, constructed using new values $\tilde{c}$, along the channel a to the observing process; typically the observers type environment Γ will be augmented with knowledge of v. This is implemented via the rules (C-OUT) and (C-OPEN), which uses the *top* type $\top$ to denote the type consisting of the empty set of capabilities; note that this dominates all channel types in the subtyping relation.

In the judgements (1) above it should not be assumed that P can be typed in Γ; indeed in the rules of Figure 5 it is not even assumed that the process terms are even typeable. Intuitively we expect the process P to be well typed in some environment Δ and that Γ represents that part of Δ which is known to the user. This motivates the following definition.

Definition 3.2 The pair $\Gamma \rhd P$ is said to be a *configuration* if there exists an environment Δ such that

- $\Delta <: \Gamma$
- $\mathrm{domain}(\Delta) = \mathrm{domain}(\Gamma)$
- $\Delta \vdash P$

One can check that configurations are preserved by the relations defined in Figure 5; in the sequel we will refer to such judgements, applied to configurations, as *actions in context*.

It is worthwhile noting that in actions in context, (1) above, the new observers environment Γ' is not in general determined by the initial configuration $\Gamma \rhd P$, unlike in [10]. This arises because of the rule (L-OUT) in Figure 5. In

general $\Gamma(a)$ may contain two read capabilities, $\mathsf{r}_{\delta_1}\langle \mathrm{A}_1 \rangle$ and $\mathsf{r}_{\delta_2}\langle \mathrm{A}_2 \rangle$, in which case Γ' may take either of the forms $\Gamma \sqcap v : \mathrm{A}_1$ or $\Gamma \sqcap v : \mathrm{A}_2$. However by restricting ourselves to single-level types this problem does not arise.

We say Γ is a *single-level* environment if it only uses single-level types. For such environments we can define the partial predicate Γ *after*$_\sigma$ s with the property that in every action in context, (1) above, if Γ is *single-level* then Γ' must be Γ *after*$_\sigma$ s; see the full version [9] for details.

It is well-known that *may testing* is related to the ability of processes to perform sequences of actions, but in an asynchronous language we must consider some form of asynchronous sequences. In the present context this means generalising *actions in context* to *(asynchronous) traces in context* :

Definition 3.3 [Traces] Let $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma \Gamma' \triangleright P'$ be the least relation which satisfies the rules given in Figure 6.

Theorem 3.4 (Alternative Characterisation of May Testing) *Suppose* $\Gamma \triangleright P$ *and* $\Gamma \triangleright Q$ *are configurations. Then* $\Gamma \triangleright_\sigma P \sqsubseteq_{may} Q$ *if and only if* $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma$ *implies* $\Gamma \triangleright Q \stackrel{s}{\Longrightarrow}_\sigma$.

The extra ingredients required to capture must testing, in addition to traces, are well-known from [15,8]; they include a *convergence predicate*, indicating that a process has no internal infinite computations, and *acceptance sets*, indicating the next possible actions in which a process can engage. Here these need to be generalised from processes to configurations; they must also be relativised to security levels.

Definition 3.5 [Convergence] We say the configuration $\mathcal{C}$ *converges*, written $\mathcal{C} \Downarrow$, if there is no infinite sequence of derivations

$$\mathcal{C} \stackrel{\tau}{\longrightarrow} \mathcal{C}_1 \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} \mathcal{C}_k \stackrel{\tau}{\longrightarrow}$$

This relation is then parameterised to *sequences in context* and security levels by

ε**:** $\mathcal{C} \Downarrow^\sigma \varepsilon$ if $\mathcal{C} \Downarrow$
$s = \alpha \cdot s'$**:** $\mathcal{C} \Downarrow^\sigma s$ if $\mathcal{C} \Downarrow$ and whenever $\mathcal{C} \stackrel{\alpha}{\Longrightarrow}_\sigma \mathcal{C}'$, $\mathcal{C}' \Downarrow^\sigma s'$.

Note that for a configuration $\Gamma \triangleright P$ whether or not it converges is actually independent of the typing environment Γ. However convergence relative to a *sequence in context* is in general dependent on these environments.

We now adapt the definition of *Acceptance sets*, [8], to the security π-calculus. First let

$$\mathcal{O}^\sigma(\mathcal{C}) = \{\, a! \mid \exists v.\mathcal{C} \stackrel{a!v}{\longrightarrow}_\sigma \,\}$$

and

$$\mathcal{R}^\sigma(\mathcal{C}) = \{\, a? \mid \exists v.\mathcal{C} \stackrel{a?v}{\longrightarrow}_\sigma \,\} \cup \mathcal{O}^\sigma(\mathcal{C}).$$

Definition 3.6 [Acceptance sets] For a configuration $\mathcal{C}$, let $A^\sigma(\mathcal{C}, s)$, its σ-

level acceptance set after s, be defined by

$$\{\, \mathcal{R}^\sigma(\mathcal{C}') \mid \mathcal{C} \stackrel{s}{\Longrightarrow}_\sigma \mathcal{C}' \not\stackrel{\tau}{\longrightarrow} \,\}$$

Similarly let its *output* acceptance set after s be given by

$$\{\, \mathcal{O}^\sigma(\mathcal{C}') \mid \mathcal{C} \stackrel{s}{\Longrightarrow}_\sigma \mathcal{C}' \not\stackrel{\tau}{\longrightarrow} \,\}$$

Note only acceptance sets from *stable* configurations, configurations $\mathcal{C}'$ such that $\mathcal{C}' \not\stackrel{\tau}{\longrightarrow}$, are used.

The security π-calculus is *asynchronous* and therefore, as explained in [3], acceptance sets are too discriminating, when used to characterise *must* testing; to see this it is sufficient to consider the simple example

$$a?(x)\,\mathbf{0} \sqsubseteq_{must} \mathbf{0}\,.$$

The same reference goes on to explain that the use of *output* acceptance sets must also be relativised to sets of input actions, which we now explain.

Input Completions. We use $I^\sigma(\mathcal{C})$ to denote the set of input actions which the configuration $\mathcal{C}$ can perform at level σ, $\{\, a?v \mid \mathcal{C} \stackrel{a?v}{\longrightarrow}_\sigma \,\}$. More generally we use I to denote an arbitrary *multi-set* of input actions, $c(I)$ to denote $\{\, a? \mid a?v \in I \,\}$ and $\overline{c}(I)$ its converse, $\{\, a! \mid a?v \in I \,\}$.

Then $\searrow_I^\sigma$ is defined to be the least relation which satisfies

- $\mathcal{C} \not\stackrel{\tau}{\longrightarrow}$ and $I^\sigma(\mathcal{C}) \cap I = \emptyset$ implies $\mathcal{C} \searrow_I^\sigma \mathcal{C}$
- $\mathcal{C} \stackrel{i}{\Longrightarrow}_\sigma \mathcal{C}'$ and $\mathcal{C}' \searrow_I^\sigma \mathcal{C}''$ implies $\mathcal{C} \searrow_{I \uplus \{i\}}^\sigma \mathcal{C}''$.

Intuitively $\mathcal{C}' \searrow_I^\sigma \mathcal{C}$" means that $\mathcal{C}$ can evolve to a stable configuration $\mathcal{C}'$ by performing a subset of the input actions in the multi-set I; moreover this subset is maximal in the sense that $\mathcal{C}'$ can not perform any of the remaining actions.

Definition 3.7 [Asynchronous acceptance sets] For a configuration $\mathcal{C}$, let $O_I^\sigma(\mathcal{C}, s)$, its σ-level asynchronous acceptance set after s, relative to the multi-set of input actions I, be defined by

$$\{\, \mathcal{O}_I^\sigma(\mathcal{C}'') \mid \mathcal{C} \stackrel{\mathcal{C}'}{\Longrightarrow}_\sigma,\ \mathcal{C}' \searrow_I^\sigma \mathcal{C}'' \,\}.$$

With one final notational convention we can mimic the alternative characterisation of must testing from [3]. We write Γ *allows*$_\sigma$ $a?v$ if $\Gamma \models_\sigma a!\langle v \rangle$; this is generalised to sets of actions in the normal manner.

Definition 3.8 Let $\mathcal{C},\ \mathcal{D}$ be configurations of the form $\Gamma \triangleright P$, $\Gamma \triangleright Q$ respectively. Then $\mathcal{C} \ll^\sigma \mathcal{D}$ if for every s,

$\mathcal{C} \Downarrow s$ implies $a)$ $\mathcal{D} \Downarrow s$

$b)$ $\forall D \in \mathcal{A}^\sigma(\mathcal{D}, s), \forall I$ such that $c(I) \cap D = \emptyset$ and $(\Gamma \; after_\sigma \; s) \; allows_\sigma \; I$, $\exists O \in \mathcal{O}^\sigma_I(\mathcal{C}, s)$ such that $O - \overline{c}(I) \subseteq D$.

Theorem 3.9 *Let Γ be a single-level environment such that both $\Gamma \triangleright P$ and $\Gamma \triangleright Q$ are configurations. Then $\Gamma \triangleright_\sigma P \sqsubseteq_{must} Q$ if and only if $\Gamma \triangleright P \ll^\sigma \Gamma \triangleright Q$.*

4 Non-Interference Results

In this section we reconsider the approach taken to non-interference in Section 4 of [11]. The essential idea is that if a process is well-typed at a given level σ then its behaviour at that level is independent of processes "running at higher security levels"; or more generally "running at security levels independent to σ". A particular formulation of such a result was given in Theorem 5.3 of [11]:

Theorem 4.1 *If $\Gamma \vdash_\sigma P, Q$ and $\Gamma \vdash_{\mathsf{top}} H, K$, where H, K are σ-*free *processes, then $\Gamma \triangleright_\sigma P \simeq_{may} Q$ implies $\Gamma \triangleright_\sigma P \mid H \simeq_{may} Q \mid K$.*

Here, because of our more refined notions of well-typing, and the characterisation result Theorem 3.4 we can give offer a significant improvement on this theorem, and moreover the formulation is actually easier.

Let us say that the security level δ is independent of σ if $\delta \not\preceq \sigma$. We can ensure that a process H is "running at a security level independent to σ" by demanding that $\Gamma \vDash^\delta H$, for some δ independent of σ. In fact we will only require the weaker typing relation $\Delta \vDash^{w\delta} H$. This ensures that all the output actions of H are at a level independent of σ, as can be deduced from the following property:

Lemma 4.2 *Suppose $\Gamma \vDash^{w\delta} H$. Then $\Gamma \triangleright H \xrightarrow{\mu}_\rho$, where μ is an output action, implies $\delta \preceq \rho$.*

Definition 4.3 We say a process H is σ-high with respect to Γ if $\Gamma \vDash^{w\delta} H$ for some δ independent of σ.

We can now state our first non-interference result. Note that it applies to processes such that $\Gamma \vdash_{r\sigma} P$, Q rather than the more restrictive $\Gamma \vdash_\sigma P$, Q, as in Theorem 4.1; only their input actions need to be at level at most σ.

Theorem 4.4 (Non-Interference 1) *Suppose $\Gamma \vdash_{r\sigma} P$, Q. Then*

$$\Gamma \triangleright^\sigma P \sqsubseteq_{may} Q \text{ implies } \Gamma \triangleright^\sigma P \mid H \sqsubseteq_{may} Q \mid K$$

for all σ-high processes H, K.

Non-interference with respect to *may* testing equivalence gives a certain level of assurance that there is no information flow from high-level processes to low-level processes. But it has been argued in [5] that the stronger the

equivalence used in the formulation of interference-freeness, the more we can be assured that high-level information can not be leaked. An example may be found in [4] where it is argued that the ability of a low-level user to observe the potential *absence* of actions may result in an undesirable flow of information. Our second non-interference result, which relies on the characterisation theorem Theorem 3.9, addresses this problem as it is expressed in terms of *must* testing equivalence; this is sensitive to potential deadlocks, and therefore the potential absence of actions.

First note that Theorem 4.4 is no longer true when $\sqsubseteq_{may}$ is replaced by $\sqsubseteq_{must}$, as the following example shows.

Example 4.5 Let A, B denote the types $\{\mathsf{w}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{top}}\langle\rangle\}$ and $\{\mathsf{r}_{\mathsf{top}}\langle\rangle\}$ respectively. Further, let Γ map a to A and n to the type $\{\mathsf{w}_{\mathsf{bot}}\langle \mathrm{A}\rangle, \mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle, \mathsf{r}_{\mathsf{top}}\langle \mathrm{B}\rangle\}$. Now consider the processes P and H defined by

$$P \Leftarrow n!\langle a\rangle \mid n?(x:\mathrm{A})\ x!\langle\rangle \qquad\qquad H \Leftarrow n?(x:\mathrm{B})\,\mathbf{0}$$

It is very easy to check that $\Gamma \vdash_{r_{\mathsf{bot}}} P$ and $\Gamma \vdash^{w_{\mathsf{top}}} H$. However

$$\Gamma; \Gamma \rhd^{\mathsf{bot}} P \mid \mathbf{0} \not\sqsubseteq_{must} P \mid H$$

because of the bot level test $a?()\ \omega!\langle\rangle$.

The presence or absence of H determines whether or not there is read contention on the channel n, which in turn influences the deadlock capabilities of P with respect to the channel a.

Here the problem is the type of the channel n; it may be read at both level bot and top. Note that such examples, where there is contention between reads at different levels, can not be expressed in the *join calculus*, [4].

A not unreasonable restriction would be to require that the read capability of channels be confined to a particular security level, using single-level types. This would not rule out inter-level communication, but simply control it more tightly. This restriction can be enforced requiring the type-checking to use single-level types and forbidding high-level processes to read from low-level channels.

Definition 4.6 We say a process H is *strong* σ-high with respect to Γ if $\Gamma \vdash^{\delta} H$ for some δ independent of σ.

Theorem 4.7 (Non-Interference 2) *Suppose* $\Gamma \vdash_{r_\sigma} P, Q$, *where* Γ *is a single-level environment. Then*

$$\Gamma \rhd^{\sigma} P \sqsubseteq_{must} Q \text{ implies } \Gamma \rhd^{\sigma} P \mid H \sqsubseteq_{must} Q \mid K$$

for all finite strong σ*-high processes* H, K.

Note that we must restrict our attention to finite H and K since *must* testing is sensitive to divergence; if H is a divergent term then we could not

expect $\Gamma \rhd^{\sigma} P \mid \mathbf{0} \simeq_{must} P \mid H$ to hold when P is a convergent term. This problem is avoided by restricting attention to finite terms, which can never diverge.

5 Related Work

A general overview of the use of static analysis techniques to enforce information-flow policies may be found in [20]. Useful surveys of research into non-interference in process languages are given in [5,19] [3]. Much of this work is behaviour based; systems are deemed to be interference-free if their trace sets, sequences of actions labelled *high* or *low*, satisfy certain properties. Here we use a more extensional approach, saying that a system is interference-free if low-level observers are unable to discern the presence or absence of high-level components. There must, of course be some connection between our definition and at least one of the behavioural definitions in the literature. However the comparison is not straightforward. The definitions, in papers such as [18,5] are for very simple untyped versions of CCS or CSP, while much of the power of our approach comes from the use of types for the more sophisticated π-calculus.

In [14] a type system is given which guarantees non-interference with respect to an extension of the π-calculus; moreover non-interference is expressed with respect to a barbed congruence. However the language used is a considerable extension of the π-calculus, with operators for selection based input/output, based on disjunctive patterns, and it is the behaviour of these operators which are mainly constrained by the type system. The types used are also very sophisticated. Unlike ours, which are simply annotated versions of the standard Pierce/Sangiorgi types, [16], they track the use of channels, using annotated affine and linear types, and capture causal relationships between actions by a partial composition on these types, using ideas based on the *graph types* of [24].

Finally [4], which uses security labels attached to messages in the join calculus to formulate non-interference, argues via an example for the use of a behavioural equivalence stronger than *may* testing. The formulation uses *weak barbed congruence* but could have equally well used *must* testing equivalence; indeed it is difficult to envisage a practical scenario in which there is something to be gained from assuming attackers have the extra power associated with the former rather than the latter. We have also already pointed out (in Example 4.5) that the join calculus can not be used to express situations in which there is read contention between different security levels. Nevertheless the approach used to develop a type system for the join calculus for detecting information flow seems to be quite general and may be applicable to the asynchronous π-calculus.

[3] For the use of types for other languages see [22].

References

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

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

[3] 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.

[4] Sylvain Conchon. Modular information flow analysis for process calculi. In Iliano Cervesato, editor, *Proceedings of the Foundations of Computer Security Workshop (FCS 2002)*, Copenhagen, Denmark, JULY 2002.

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

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

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

[8] M. Hennessy. *An Algebraic Theory of Processes*. MIT Press, 1988.

[9] Matthew Hennessy. The security picalculus and non-interference. Technical report 2000:05, University of Sussex, 2000. Available from `http://www.cogs.susx.ac.uk/`.

[10] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. In James Harland, editor, *Electronic Notes in Theoretical Computer Science*, volume 61. Elsevier Science Publishers, 2002.

[11] 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, July 2000. Springer-Verlag. Full version to appear in ACM Transactions on Programming Languages and Systems.

[12] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[13] 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.

[14] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *29th Annual Symposium on Principles of Programming Languages*. ACM, January 2002.

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

[16] B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(3):531–584, 2000.

[17] 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.

[18] 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.

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

[20] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communication*, 2002. To appear.

[21] D. Sangiorgi and D. Walker. *The Picalculus*. Cambridge University Press, 2001.

[22] 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.

[23] David Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. Ph.d. thesis, Edinburgh University, 1995.

[24] Nobuko Yoshida. Graph types for monadic mobile processes. In *FSTTCS*, volume 1180, pages 371–386. Springer-Verlag, 1996.