may be transmitted on channels includes channel names themselves; this, together with the ability to dynamically create new channel names, gives the language its descriptive power.

Within the setting of the $\pi$-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, $\sigma[\![$

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 $\mathsf{hl}$ is a channel with high-level write access and low-level read access in:

$$\mathsf{top}\,[\![\, h?(x)\,\mathsf{if}\; x = 0\;\mathsf{th}\;\;\mathsf{n}\;\;\mathsf{hl}!\langle 0\rangle \quad \mathsf{ls} \quad \mathsf{hl}!\langle 1\rangle \,]\!] \mid \mathsf{bot}\,[\![\, \mathsf{hl}?(z) \quad ]\!]$$

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 .

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 $\pi$-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 $\sigma$ may only write to channels at level $\sigma$ or above
- *read down*: a process at level $\sigma$ may only read from channels at level $\sigma$ 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-interfer*

FIGURE 2 Labelled Transition Semantics

(L-OUT) (L-IN)

$$\overline{a!\langle v\rangle \xrightarrow{a!v} \mathbf{0}} \qquad \overline{a?(X)\ P \xrightarrow{(\widetilde{c}:\widetilde{\mathrm{C}})a?v} P\{\!|{}^{v}/_{X}|\!\}}\ \tilde{c} \notin \mathsf{fn}(P)$$

(L-OPEN)

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

(L-COM)

$$\frac{P \xrightarrow{\alpha} P', \qquad \xrightarrow{\overline{\alpha}} \quad '}{P \mid \quad \xrightarrow{\tau} (\mathsf{n}\ \mathsf{w}\, \mathcal{E}(\alpha))\ (P' \mid \quad ')}$$

(L-EQ)

$$\overline{\mathsf{if}\ u = u\ \mathsf{th}\ \mathsf{n}\ P\ \ \mathsf{ls} \quad \xrightarrow{\tau} P} \qquad \overline{\mathsf{if}\ u = w\ \mathsf{th}\ \mathsf{n}\ P\ \ \mathsf{ls} \quad \xrightarrow{\tau}}\ u \neq w$$

(L-CTXT)

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

$$\sigma[\![P]\!] \xrightarrow{\mu} \sigma[\![P']\!]$$

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

a set of *basic values* $BV_\sigma$; we use $bv$ to range over base values. We require that all syntactic sets be disjoint.

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

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

this end, *Pre-capabilities* and *pre-types* are defined as follows:

| *cap* ::= | *Pre-Capability* |
|---|---|
| $\mathsf{w}_\sigma\langle \mathrm{A} \rangle$ | $\sigma$-level process can write values with type A |
| $\mathsf{r}_\sigma\langle \mathrm{A} \rangle$ | $\sigma$-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, $\{cap\}$,

FIGURE 3 Runtime Errors

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

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

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

(E-STR) $\dfrac{P \stackrel{\Sigma}{\longmapsto} err}{P \mid \quad \stackrel{\Sigma}{\longmapsto} err}$ $\dfrac{P \stackrel{\Sigma}{\longmapsto} err}{\rho[\![P]\!] \stackrel{\Sigma}{\longmapsto} err}$ $\dfrac{P \equiv \quad , \; P \stackrel{\Sigma}{\longmapsto} err}{\stackrel{\Sigma}{\longmapsto} err}$

$$\frac{P \stackrel{\Sigma, a:\mathrm{A}}{\longmapsto} err}{(\mathsf{n}\ \mathsf{w}\, n:\mathrm{A})\ P \stackrel{\Sigma}{\longmapsto} err}$$

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

## 3 Resou ce Cont ol

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

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

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

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

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

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

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

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

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 $RType_{\mathsf{bot}}$. Note also that there is no relation between subtyping and accessibility at a given security level. For example:

$$\mathsf{w}_{\mathsf{bot}}\langle\rangle \in RType_{\mathsf{bot}} \text{ and } \{\mathsf{w}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{top}}\langle\rangle\} <: \mathsf{r}_{\mathsf{bot}}\langle\rangle \text{ but } \{\mathsf{w}_{\mathsf{bot}}\langle\rangle, \mathsf{w}_{\mathsf{top}}\langle\rangle\} \notin RType_{\mathsf{bot}}$$
$$\mathsf{r}_{\mathsf{bot}}\langle\rangle \in RType_{\mathsf{bot}} \text{ and } \mathsf{r}_{\mathsf{bot}}\langle\rangle <: \mathsf{r}_{\mathsf{top}}\langle\rangle \text{ but } \mathsf{r}_{\mathsf{top}}\langle\rangle \notin RType_{\mathsf{bot}}$$

The compatibility requirement between read and write capabilities in a type (RT-WRRD), in addition to the typing implications discussed in [23], 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 $\sigma$ is $\mathsf{bot}$, 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.

*Remark.* Most of the restrictions imposed on types are essential to achieving Subject Reduction, but a few are not. First, Subject Reduction still holds if we weaken (U-WR) to: $\mathsf{w}_\sigma\langle \mathrm{A}\rangle <: \mathsf{w}_\rho\langle \mathrm{B}\rangle$ if $\mathrm{B} <: \mathrm{A}$ and $\sigma \preceq \rho$. Were we to adopt this rule, it would be true that every process typable at level $\sigma$ would also be typable at level $\rho$, for $\sigma \preceq \rho$. Given our definition, this is not true. Nonetheless, every process typable at $\sigma$ can be trivially rewritten so that it is typable at $\rho$ given our definition (one must simply surround output actions with explicit security restrictions). We have

FIGURE 4 Typing Rules

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

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

$$\frac{\Gamma \vdash u : \mathrm{A}, v : \mathrm{B} \quad \Gamma \models^\sigma \quad \Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \models^\sigma P}{\Gamma \models^\sigma \mathsf{if}\ u = v\ \mathsf{th}\ \mathsf{n}\ P\ \ \mathsf{ls}} \text{(T-EQ)}$$

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

adopted the stronger rule because it is necessary in the next section and results in no substantive loss of expressivity.

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

PROPOSITION 3.2. *For every $\rho$, $\mathrm{RType}_\rho$ is a preorder with respect to $<:$, with both a partial meet operation $\sqcap$ and a partial join $\sqcup$.*

*Proof.* Straightforward adaptation of Proposition 6.2 of [23]. The partial operations $\sqcap$ and $\sqcup$ are first defined by structural induction on types. Typical clauses are

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

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

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

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

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

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

Intuitively '$\Gamma \models^{\sigma} P$' indicates that the process $P$ will not cause any security errors if executed with security clearance $\sigma$. The rules are very similar to those used in papers such as [23, 21] for the standard IO typing of the $\pi$-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. One might expect that the judgements for values would need to ensure that a value written to a channel be accessible at the appropriate security level. This job, however, is already handled by our definition of types. For example, in order for $\mathsf{w}_{\sigma}\langle \mathrm{A} \rangle$ to be a type, A must be a type accessible to $\sigma$.

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

PROPOSITION 3.3.

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

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

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

- $\Gamma \vdash$

(T-STR), followed by (T-NEW), gives the required $\Gamma \models^{\sigma} (\mathsf{n}\ \mathsf{w}\,\mathcal{E}(\alpha))\ (P' \mid \ \ ')$. □

We can now prove the first main result:

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

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

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

We end this section with a brief discussion on the use of the syntax $\sigma[\![P]\!]$ in our language. We have primarily introduced it in order to discuss typing issues. Having defined our typing system we may now view $\sigma[\![P]\!]$ simply as notation for the fact that, relative to the current typing environment $\Gamma$, the process $P$ is well-typed at level $\sigma$, 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 $\Gamma$ such that $\Gamma \models^{\sigma} P$. This will be formalised in Section 5.

# 4 Info mation Flow

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

$$\mathsf{top}[\![h?(x)\ \mathsf{if}\ x = 0\ \mathsf{th}\ \mathsf{n}\ \mathsf{hl}!\langle 0\rangle \quad \mathsf{ls} \quad \mathsf{hl}!\langle 1\rangle\,]\!] \mid \mathsf{bot}[\![\,\mathsf{hl}?(z) \quad ]\!] \qquad (\star)$$

executing in an environment in which $h$ is a top-level read/write channel and hl is a top-level write and bot-level read channel. This system can be well-typed, using *R-types*, so the processes only access resources appropriate to their security level. Nevertheless there is some implicit flow of

$\mathsf{top} \llbracket h?(x)\,\mathsf{if}\ x = 0\ \mathsf{th}\ \mathsf{n}\ \mathsf{bot}$

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

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

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

It is very easy to check that $\Gamma \Vdash P, H$ and that $H$ is $\mathsf{bot}$-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*, [18], or *must testing*, [20], 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 [20], $P \mid \mathbf{0}$ *guarantees* the test $\mathsf{bot}[\![a?(x)\,\omega!\langle\rangle]\!]$ whereas $P \mid H$ does not.

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

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

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

In the following section, we prove a non-interference result using may testing on processes typable using *I-types*.

## 5 Noninte fe ence up to May Testing

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

DEFINITION 5.1. We write $T\Downarrow$ if $T \xrightarrow{\tau}{}^* T'$, where $T'$ has the form $(\mathsf{n}\ \mathsf{w}\,\tilde{c})\ (\omega!\langle\rangle \mid T'')$ for some $T''$ and $\tilde{c}$. □

FIGURE 5 Context LTS

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

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

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

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

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

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

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

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

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

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

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

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

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

Definition 5.7 (Traces). Let $\Gamma \triangleright P \stackrel{s}{\Longrightarrow}_\sigma \Gamma' \triangleright P'$ be the least relation such that:

(TR-$\epsilon$)

$$\frac{}{\Gamma \triangleright P \stackrel{\epsilon}{\Longrightarrow}_\sigma}$$

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

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

□

The ability to compose asynchronous traces depends on the fact that

- $\rho[\![P]\!] \xrightarrow{(\tilde{c}:\tilde{C}a!v)} \rho[\![P']\!]$ because $P \xrightarrow{(\tilde{c}:\tilde{C}a!v)} P'$.
  $\Gamma \Vdash^{\sigma} \rho[\![P]\!]$ implies $\Gamma \Vdash^{\sigma\sqcap\rho} P$ and so by induction

$$P \equiv_{\Gamma} (\mathsf{n}\ \mathsf{w}\,\tilde{c}:\tilde{\mathrm{C}})\ (\delta[\![a!\langle v\rangle]\!] \mid P')$$

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

□

Proposition 5.12 (Trace Composition). *Suppose* $\Gamma \Vdash^{\sigma} T$.

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

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

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

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

    Now we may apply induction to $\Gamma, \tilde{c}{:}\tilde{C}(P' \mid H') \triangleright \stackrel{s_c}{\Longrightarrow}{}^{a}_{\sigma}$, since $\Gamma \Vdash^{\sigma} P'$ by Subject Reduction and to $\Gamma \not\vdash$

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

□

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

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

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

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

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

## 6 Conclusions and Related Wo k

In this paper we have proposed simple typing systems for enforcing a variety of security properties for the *security* $\pi$*-calculus*. The types are obtained by adding security levels to the standard input/output types of the $\pi$-calculus, [21, 23]. The first typing system, based on *R-Types*, is designed with resource access control in mind; the security level of a resource (or more formally a capability on a resource) dictates the security oh earanceP

a high-level process reads a value from a low-level channel

ticated type system is used to control information flow. The judgements in their system take

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

[3] C. Bodei, P. Degano, F. Nielson, and H. . Nielson. Control flow analysis for the $\pi$-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. . 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 $\pi$-calculus. Technical eport 1702, IN IA-Sophia Antipolis, 1992.

[6] Ilaria Castellani and Matthew Hennessy. Testing theories for asynchronous languages. In V Arvind and amanujam, editors, *18th Conference on Foundations of Software Technolog 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] iccardo Focardi, Anna Ghelli, and oberto Gorrieri. Using non interference for the analysis of security protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Securit Protocols*, 1997.

[9] iccardo Focardi and oberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Securit* , 3(1), 1995.

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

[11] iccardo Focardi and oberto 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. emy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *CONCUR: Proceedings of the International Conference on Concurrenc Theor* , volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, August 1996. Springer-Verlag.

[13] . eitmas G. Andrews. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and S stems*, 2(1):56–76, 1980.

[14] J. A. Goguen and J. Meseguer. Security

*retical Computer Science*, 114:149–171, 1993.

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

[21] 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.

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

[23] James iely and Matthew Hennessy. esource 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 eport 2/98, University of Sussex, 1997. Available from `http://www.cogs.susx.ac.uk/`.

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

[25] A.W. oscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through determinism. In *European S mposium on Research in Computer Securit* , volume 875 of *LNCS*, 1994.

[26] P.Y.A. yan 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 S mposium on Principles of Programming Languages*, San Diego, January 1998.

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