Note on a simple type system for non-interference

Steffen van Bakel and Maria Grazia Vigliotti

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK

Abstract

We consider CCS with value passing and elaborate a notion of noninterference for the process calculi, which matches closely that of the programming language. The idea is to view channels as information carriers rather than as "events", so that emitting a secret on output channel can considered safe, while inputting a secret may lead to some kind of leakage. This is in contrast with the standard notion of noninterference for the process calculi where any causal dependency of low-level action from any high-level action is forbidden.

Key words: Process algebra, non-interference, type system, security policies.

Introduction

In recent years *secure information flow* has attracted a great deal of interest, spurred on by the spreading of mobile devices and nomadic computation, and has been studied in some depth for both programming languages and process calculi. In this paper we shall speak of the "language-based approach" when referring to programming languages and of the "process-algebraic approach" when referring to process calculi.

The language-based approach is concerned with the avoidance of secret information leakage or corruption through the execution of programs, i.e. with the security properties of *confidentiality* and *integrity*. The property of confidentiality, which appears to be the most studied, is usually formalised via the notion of *non-interference*, meaning that secret inputs should not have an effect on public outputs, since this could allow -in principle- a public user to reconstruct sensitive information. Non-interference may be

[1] Email: `svb@doc.ic.ac.uk`, `mgv98@doc.ic.ac.uk`

achieved in various ways: via program analysis, type systems, using semantics equivalencies, the implementation of security policies, etc. In most cases the languages are equipped with a type system or some other tool to enforce the compliance of programs to the desired security property.

In the process-algebraic approach the focus is on the notion of *external observer*, who ideally has nothing to do with the specification and implementation of a given system, and should not be able to infer any secret by interacting with it. The process-algebraic approach is concerned with secret events not being revealed while processes communicate, i.e. actions that involve sensitive or confidential data should have no effect on public actions.

Also, in process algebra, many non-interference properties are formalised in a way similar to programming languages, i.e. using program analysis, type systems, using semantics equivalencies. In the last few years a variety of properties have been proposed for process calculi, mostly based on trace equivalence or bisimulation, ranging from the simple property of *Non-deducibility on Composition* to more complicated ones (see [3] for a review).

Methods for static detection of insecure processes have not been largely studied for process calculi. In [6,7] type systems which characterise a non-inference property have been proposed for the π-calculus. More sophisticated type systems have been extensively studied in [8,9] for variants of the π-calculus, which combine the control of security with other correctness concerns. More recently, Crafa and Rossi proposed in [2] a simple security type system for the π-calculus, which consists essentially of a simplification of that used by Hennessy [7], ensuring the absence of explicit information flows. All those type systems include specific analysis on the values passed on a channel.

Pottier [11] proposed a very simple view on non-interference via a type system for the π-calculus which does not involve any extra typing information on the values passed over channels. The great appeal of this type system is its simplicity in characterising non-interference only; in fact, Pottier calls this system 'simple', and we will use his terminology in this paper. The limitation of Pottier's work, with respect to the 'simple type system' is the lack of a robust semantic notion of non-interference. In this paper we will address this issue specifically.

In process algebraic approach, differently from language based security, no distinction is made between input events and output events, neither at the level of semantics definitions of security not at the level of type systems. In this paper we aim to address two issues:

(i) study the relationship between those type systems and the *semantics-based approach* in process calculi [3,5,4];

(ii) to define a notion of non-interference which matches closely the one in the language-based approach. In other words, the basic idea is to view channels as information carriers, so that emitting a secret on an output channel can be considered safe, while inputting a secret may lead to some kind of leakage.

As for the first issue, the notion of *Persistent Non-deducibility on Composition* developed for CCS [5,4] has shown to be quite natural, also because it preserves the notion of non-interference of the language-based approach in the process-algebraic approach [5]. In this paper we will show that the 'simple type system' can be adapted to standard CCS and that it characterises the semantic notion of Persistent Non-deducibility on Composition. This means that any typeable process is persistently deducible on composition. We will show that there exist processes that are considered secure according the notion of persistence, yet that are not typeable. Therefore, the set of typeable processes according to Pottier's type system is strictly smaller than the class of processes included in Persistent Non-deducibility on Composition relation.

We consider CCS here instead of the π-calculus because we wish to focus on the specific issues of non-interference in the simplest model possible. It is clear that our work could be easily extended to the π-calculus, with little extra effort. As for the second issue, we modify the 'simple type system' so that the notion of non-interference matches closely that of programming languages. That is to view channels as information carriers rather than as "events", so that the process $a_h(x).\overline{b_l}\langle e\rangle$, which emits on a low channel a value received on a high channel, is considered insecure, while $\overline{a_h}\langle v\rangle.\overline{b_l}\langle v\rangle$, which emits successively a value v on a high channel and on a low channel, is considered secure. The second example would not be be typeable in the 'simple type system' nor would it be considered secure with the standard semantic notions of non-interference.

The rest of the paper is organised as follows: in section 1 we introduce CCS; in section 2 we introduce the notion of equivalence-based security; in section 3 we adapt the simple type system to CCS and we show that the every typeable process is secure according to the Persistent Non-deducibility on Composition. Finally, in section 4 we introduce our refined type system and elaborate on a semantics notion of non-interference based on the idea that only high-level inputs are critical for the definition of non-interference. Conclusions follow.

1 CCS

We will consider a variant of CCS with value passing, with two main differences from standard presentation:

(i) We assume the existence of a lattice $(\mathcal{L}, \leq)$, which expresses the security level of channels. Greek letters $\sigma, \tau, \rho \ldots$ and ℓ range over $\mathcal{L}$. The language CCS we consider is typed in the sense that we explicitly incorporate the security level of the channel in the syntax of the language.

(ii) We consider the value passing CCS -though value passing could be encoded with infinite choice operator [10]- without if-then-else operator as in [5]. We prefer to consider CCS with value passing in order to emphasise the different role of input and output; yet the if-then-else operator can be encoded in CCS [10] and therefore is not essential in the current presentation.

Definition 1.1 Let $\mathcal{N}$ be a enumerable set of names and $\overline{\mathcal{N}}$ an enumerable set of conames. We use the usual conventions for input $a(x)$ and output $\overline{a}\langle e\rangle$. The enumerable set of *variables* is ranged over by $x, y, z\ldots$, and the set of *values* $\mathcal{V}$ is ranged over by e; we will assume that $(\mathcal{N} \cup \overline{\mathcal{N}}) \cap \mathcal{V} = \emptyset$.

The syntax of (typed) *process prefixes*, ranged over by α, β, γ, is given by:

$$\alpha ::= a_\ell(x) \mid \overline{a_\ell}\langle e\rangle$$

where ℓ is taken from a lattice $(\mathcal{L}, \leq)$of security levels.

The set $\mathcal{P}r$ of processes, ranged over by P, Q, is given by the grammar:

$$P, Q ::= \mathbf{0} \mid \textstyle\sum_{i\in I}\alpha_i.P_i \mid P|Q \mid (\nu a_\ell)\,P \mid A[\vec{e}].$$

where I is a finite index set.

The informal meaning of process is standard: *choice operator* $\sum_{i\in I}\alpha_i.P_i$ represents the non-deterministic choice among different processes; *parallel composition* $P|Q$ represent processes running together, possibly in an interleaving fashion; *restriction* $(\nu a_\ell)\,P$ makes the name a_ℓ local to the process P.

Definition 1.2 [Notions and Conventions]

- The notion of free and bound names in $P$is standard, taking into account that $(\nu a)\,P$ is the only binding operator. With $n(P)$ we mean the set of names in P.
- For an *Agent* $A[\vec{a}]$ we assume the existence of identifier A such that a process P can be associated to that identifier, written $A[\vec{x}] \triangleq P$ when $fn(P) \subseteq \{x_1, \ldots x_n\}$.
- We assume that prefixes with the same channel name have the same security level i.e. if $a_\ell(x).P$ and $\overline{a_{\ell'}}\langle e\rangle.Q$ then $\ell = \ell'$.
- We write $P\{e/x\}$ $(P\{a/x\})$ for the standard replacement of every occurrence of x in P by the value e (the name a).
- An element of the set of actions Act is defined as $Act \triangleq \{ae \mid a \in \mathcal{N} \cup \overline{\mathcal{N}},\, e \in \mathcal{V}\} \cup \{\tau\}$; the Greek letters $\alpha, \beta\ldots$ will range over Act.
- We define $subj(a_\ell(x)) = a_\ell = subj(\overline{a_\ell}\langle e\rangle)$ and $subj(\tau) = \tau$.

Definition 1.3 [Operational Semantics] The relation $\longrightarrow \subseteq \mathcal{P}r \times Act \times \mathcal{P}r$, written $P \xrightarrow{\alpha} P'$, is defined by:

$$(\textsc{Input}) : \frac{}{\sum_{i\in I}\alpha_i.P_i \xrightarrow{ae} P_j\{e/x\}}\ (\alpha_j = a_\ell(x)) \qquad (\textsc{Par Left}) : \frac{P \xrightarrow{\alpha} P'}{P\,|\,Q \xrightarrow{\alpha} P'\,|\,Q}$$

$$(\textsc{Output}) : \frac{}{\sum_{i\in I}\alpha_i.P_i \xrightarrow{\overline{a}e} P_j}\ (\alpha_j = \overline{a_\ell}\langle e\rangle) \qquad (\textsc{Par Right}\ ) : \frac{P \xrightarrow{\alpha} P'}{Q\,|\,P \xrightarrow{\alpha} Q\,|\,P'}$$

$$(\textsc{Restr}) : \frac{P \xrightarrow{\alpha} P'}{(\nu b)P \xrightarrow{\alpha} (\nu b)P'}\ (b \neq subj(\alpha)) \qquad (\textsc{Par Comm}_1) : \frac{P \xrightarrow{ae} P' \quad Q \xrightarrow{\overline{a}e} Q'}{P\,|\,Q \xrightarrow{\tau} P'\,|\,Q'}$$

$$(\textsc{Rec}) : \frac{P\{\overrightarrow{b/x}\} \xrightarrow{\alpha} P'}{A[\vec{b}] \xrightarrow{\alpha} P'}\ (P \triangleq A(\vec{x})) \qquad (\textsc{Par Comm}_2) : \frac{P \xrightarrow{\overline{a}e} P' \quad Q \xrightarrow{ae} Q'}{P\,|\,Q \xrightarrow{\tau} P'\,|\,Q'}$$

We adopt the usual notational conventions. We write $\xrightarrow{\tau}{}^*$ for the reflexive and transitive closure of $\xrightarrow{\tau}$. We define define $P \overset{\alpha}{\Rightarrow} P'$ as $P \xrightarrow{\tau}{}^* \xrightarrow{\alpha} \xrightarrow{\tau}{}^* P'$ and $P \overset{\hat{\alpha}}{\Rightarrow} P'$ as $P \overset{\alpha}{\Rightarrow} P'$ if $\alpha \neq \tau$ or $P \xrightarrow{\tau}{}^* P'$ otherwise. Thus $P \overset{\tau}{\Rightarrow} P'$ requires at least one τ-transition while $P \overset{\hat{\tau}}{\Rightarrow} P'$ allows for the empty move.

2 Equivalence-based security

In this section, we shall examine previous definitions of equivalence-based security that aim to capture the notions of non-interference. There are many different definitions, based on semantics equivalencies [3]. We consider in this paper only non-interference bisimilarity for two reasons: (1) these equivalencies are very common in the literature [2,11,4,5,1], etc, and (2) there are well-established proof-methods to show when processes are equivalent. We shall first consider Bisimulation-based Non-Deducibility on Compositions (BNDC) followed by Persistent Bisimulation-based Non-Deducibility on Compositions (P-BNDC).

Definition 2.1 [Weak Bisimulation] A symmetric binary relation $\mathcal{S} \subseteq \mathcal{P}r \times \mathcal{P}r$ is a *weak bisimulation* if $P\mathcal{S}Q$ implies, for all $a \in Act$:

- whenever $P \xrightarrow{\alpha} P'$ then there exists a Q' such that $Q \overset{\hat{\alpha}}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$.

Two processes P and Q are *weakly bisimilar*, written $P \approx Q$, if for some weak bisimulation $\mathcal{S}$, $P\mathcal{S}Q$.

It is well known that $\approx$ is both the largest bisimulation and an equivalence relation.

In this section, we will assume –without loss of generality– that the lattice of security levels $\mathcal{L}$ will be simply $\{l,h\}$, with $l \leq h$, where l stands for "low" or "public", and h stands for "high" or "secret" as also done in

[4,5,1]. The current work could be extended to a more general notion of lattice, however we argue that from a semantics and security point of view a more general notion of lattice would not give more expressiveness. In fact, in the semantics definition of non-interference we express the fact that public action cannot have any form of casual dependency from secret actions. This means that in a general lattice, actions below a certain security level are considered of public domain, and all the action above a given security level must be protected. That means in actual fact that it is sufficient to consider a collapsed lattice with two security levels only.

Before proceeding to the definition of the security relation we fix some notation.

Definition 2.2 [Notation]

- We write $\mathcal{P}r_H$ for the subset of process that have prefixes with type h only.
- We write $(\nu A)\ P$ where A is a *set of names* for the restriction in P of all the names present in A.
- We write $(\nu\mathcal{H})\ P$ to to mean that we restrict *all* the names that have security level h in the process P.

The first definition of equivalence-based non interference uses the definition of weak bisimilarity directly.

Definition 2.3 [BNDC] Let P be a process. P is said to be *secure*, $P \in \mathsf{BNDC}$, if for every process $\Pi \in \mathcal{P}r_H$, $(\nu\mathcal{H})(P \mid \Pi) \approx (\nu\mathcal{H})P$.

The BNDC requires that high level actions present in the process Π have have no effect on the execution of P.

Clearly any process P which does not contain high names is secure. In fact, we have on one side $(\nu\mathcal{H})(P \mid \Pi) \approx P \mid (\nu\mathcal{H})(\Pi)$ where $(\nu\mathcal{H})(\Pi) \approx \mathbf{0}$, and on the other side $(\nu\mathcal{H})P \approx P$. Any process P which contains only high names is secure, since all processes can only perform τ actions. Insecurity may appear when a high name is sequentially followed by a low name in P, because in this case the execution of $(\nu\mathcal{H})P$ may block on the high name (if this is reachable), making the low name unreachable, while it is always possible to find a high process Π that makes the low name reachable in $(\nu\mathcal{H})(P \mid \Pi)$. Typical examples of insecure processes of this kind are $a_h(x).\overline{b_l}\langle e\rangle$ and $a_h(e).\overline{b_l}\langle e\rangle$. These examples show that the BNDC does not distinguish whether a low level action comes after an input or an output. Quite surprisingly, insecurity appears when a high name is in conflict with a low name in P, that is, when they occur in different branches of a choice, as in the process $a_h(x) + \overline{b_l}\langle e\rangle$. It is disputable if this process should be considered insecure since the low and high level actions are independent. Finally, the process: $a_h(x).\overline{b_l}\langle e\rangle + \overline{b_l}\langle e\rangle$ is secure.

As argued in [4,5], Bisimulation-based Non-Deducibility on Compositions is not strong enough to deal with dynamic contexts. A strengthening of this notion, called Persistent Bisimulation-based Non-Deducibility on Compositions (P-BNDC) was therefore proposed in [4]. We shall adopt this notion as the starting point for our study.

To define P-BNDC, a new kind of transition $\overset{\widetilde{\alpha}}{\Rightarrow}$ is introduced, defined as follows for any $\alpha \in Act$.

Definition 2.4 The relation $\overset{\widetilde{\alpha}}{\Rightarrow}$ is defined as $\overset{\hat{\alpha}}{\Rightarrow} \cup \xrightarrow{\tau}^{*}$ when $subj(\alpha) \in \mathcal{H}$, or in the usual manner when a is a low level action.

The definition of weak bi-simulation up-to-high used the new relation in the definition.

Definition 2.5 [Weak bi-simulation up-to-high] A symmetric binary relation $\mathcal{S} \subseteq \mathcal{P}r \times \mathcal{P}r$ is a *weak bisimulation up-to-high* if an only if $P\mathcal{S}Q$ implies that, for all $a \in Act$:

- whenever $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \overset{\widetilde{\alpha}}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$.

Two processes P, Q are *weakly bisimilar up-to-high*, written $P \approx_{\mathcal{H}} Q$, if $P\mathcal{S}Q$ for some weak bisimulation up-to-high $\mathcal{S}$.

In other words, when a process makes a high-level action, could be matched by any number of τ-action. This definition abstracts away from high level actions.

Definition 2.6 [P-BNDC] P is said persistently secure, $P \in$ P-BNDC if $(\nu\mathcal{H})P \approx_{\mathcal{H}} P$.

It has been shown in [4,5] that P-BNDC is strictly stronger than BNDC i.e.P-BNDC $\subset$ BNDC. In fact, if P is in the P-BNDC amounts to requiring BNDC for all reachable states of P; this explains why it is called "persistent". The example considered above for BNDC are also persistently secure; however the process:

$$a_h(v).a_h(v).\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle$$

is secure but not persistently secure.

3 A simple type system

In this section we will adapt the type system as developed by Pottier [11] for the π-calculus, to CCS. That type system was devised with the idea of defining the simplest possible types that would guarantee non-interference. In that paper, Pottier works mostly with the π-calculus with replication and general choice. In particular, we simplify the original type system and we adapt the rule of replication to recursion and eliminate the rule (NORM) used to guarantee that all the prefixes in the choice have the same security level. Because the version of CCS used here has only guarded choice, the rule (Norm) is not longer necessary.

We will now introduce the type system: it assigns *security levels* to channels in processes.

Security levels are elements σ, τ of a lattice $(\mathcal{L}, \leq)$: a flow from level σ to level τ is authorised if and only if $\sigma \leq \tau$. We use $\sqcap$ and $\sqcup$ for the operations of, respectively, meet and join on this lattice.

Type judgements for processes have then the form $\Gamma \vdash P : \ell$ which informally means that the process P can be inferred from the environment Γ at security level ℓ, where ℓ is a meta-variable ranging over the security lattice.

Definition 3.1 [Type Assignment] A *type environment* Γ is a mapping from channel names to security levels such that $\Gamma(a) = \Gamma(\overline{a})$; we write $a{:}\ell \in \Gamma$ whenever $\Gamma(a) = \ell$. We naturally extend the mapping to prefixes α by $\Gamma(\alpha) = subj(\alpha)$.

The assignment of (security) types to processes is defined via the following natural deduction system.

$$(\textsc{Nil}) : \frac{}{\Gamma \vdash \mathbf{0} : \ell} \qquad (\textsc{Rec}) : \frac{\Gamma, x_1{:}\ell_1, \ldots, x_n{:}\ell_n \vdash P : \ell}{\Gamma, b_1{:}\ell_1, \ldots, b_n{:}\ell_n \vdash A[\vec{b}] : \ell}\ (A[\vec{x}] = P)$$

$$(\textsc{Sub}) : \frac{\Gamma \vdash P : \ell}{\Gamma \vdash P : \ell'}\ (\ell' \leq \ell) \qquad (\textsc{Sum}) : \frac{\Gamma \vdash \alpha_i.P_i : \ell \quad \Gamma(\alpha_i) = \ell \quad (\forall i \in I)}{\Gamma \vdash \sum_{i \in I} \alpha_i.P_i : \ell}$$

$$(\textsc{Comp}) : \frac{\Gamma \vdash P : \ell \quad \Gamma \vdash Q : \ell}{\Gamma \vdash P | Q : \ell} \qquad (\textsc{Restr}) : \frac{\Gamma, a{:}\ell' \vdash P : \ell}{\Gamma \vdash (\nu a_{\ell'})\, P : \ell}$$

Definition 3.2 P is *typeable* if $\Gamma \vdash P : \ell$ for some Γ and ℓ.

Clearly not all processes are typeable. For instance $a_h(x).\overline{b_h}\langle e \rangle$ is not typeable. Here the difference between the type system and the general typed language as defined in this paper is made clear. The type language does not impose any constraint on the construction of processes. Thus, the process $a_h(x).\overline{b_h}\langle e \rangle$ is a legal term according to our syntax, but it is not possible to find an environment Γ such that will assign to the process $a_h(x).\overline{b_h}\langle e \rangle$ a type ℓ.

The following theorem states that no matter how the process behaves, there will be no leakage of sensitive data, since types are preserved by reductions.

Theorem 3.3 (Subject reduction) *If* $\Gamma \vdash P : \ell$ *and* $P \xrightarrow{\alpha} P'$ *then* $\Gamma \vdash P' : \ell$.

Proof. By induction on the inference of $\Gamma \vdash P : \ell$. □

In this section we analyse the relationship between the 'simple type system' developed by Pottier [11] and P-BNDC [5]. We shall see that every typeable process according to Pottier's type system is secure according to the P-BNDC.

Proof. By induction on the inference of $\Gamma \vdash P : \ell$. $\square$

Theorem 3.4 *If P is typeable, then $P \in$ P-BNDC.*

The reverse of the above theorem is not true. In fact, $a_h.b_l + b_l \in$ P-BNDC while this process cannot be typed in the type system above. We conclude that if P is typeable then it is persistently secure and secure. By the examples presented in this paper, not all secure processes are typeable or persistently secure.

Also $\approx_{\mathcal{H}}$ is not preserved by parallel composition on arbitrary programs, as shown by the following example where $P_i \approx_{\mathcal{H}} Q_i$ for $i = 1,2$ but $P_1 \mid P_2 \not\approx_{\mathcal{H}} Q_1 \mid Q_2$. Take

$$P_1 = a_h(x) \quad Q_1 = \mathbf{0} \quad P_2 = Q_2 = (\nu \overline{b_h}\langle e\rangle \mid b_h(x))(\overline{c_l}\langle e'\rangle . + \overline{a_h}\langle e''\rangle).$$

Clearly $\overline{c_l}\langle e'\rangle + \overline{a_h}\langle e''\rangle$ is not typeable since in the sum only prefixes at the same security level are allowed. This means that for untyped processes the P-BNDC is not closed under arbitrary contexts, which makes compositional reasoning quite difficult. It is an open question –which we leave for future work– whether P-BNDC is closed under typed contexts.

In this section we have shown that the 'simple type system ' has a natural correspondence in the P-BNDC. A type system gives an automatic way to guarantee the bsence of leakage in programs. This is the main advantage of the type system over semantics based notions of non-interference.

4 Asymmetric type system for CCS

The 'simple type system' imposes as security discipline such after high level action only low-level actions can follow. In other words, the type systems guarantees that there is not causal dependency from high level action to low-level actions. We argue that there is a difference between the action performed by an input and an output. Consider the example of two systems, where the first one simply emits signals of acknowledgements to both high and low.

$$P(ack) = \overline{ack_h}\langle e\rangle . \overline{ack_l}\langle e'\rangle . P$$

The second system is a system that first reads from a secret database and then outputs the outcome.

$$Q(ack) = read(x) . \overline{wait_l}\langle .e\rangle . \overline{write_l}\langle x\rangle . P$$

Clearly for P is makes no difference in which order the high-level and the low-level actions take place. In no way ack_l can reveal anything about ack_h

since the value of the outputs are independent. However, the situation is radically different for Q. After an high-level input, information can be *leaked* to an insecure level via a low-level output as defined in Q. Therefore, it is vital that after a high input, a low level output action is not permitted. The type system we present in the next section is a refinement of the simple type system, and distinguishes between input and output. It allows low-level actions after a high-level output under the assumption that high level outputs are not sensitive actions. On the other end, it not possible to perform a low-level action after an input as in the simple type system.

The types developed in this section are inspired by those of [1]: they record both the *reading level* of processes (as the maximal level of their input channels) and their *writing level* (the minimal level of their output channels). In other words the *types* are tuples (σ, τ), where σ stands for the security level for input and τ stand for the security level for outputs.

(i)

(ii) Type judgements for processes have the form $\Gamma \vdash P : (\sigma, \tau)$, where σ is an *upper bound* for the level of input channels of P, and τ is a *lower bound* for the level of its output channels.

Notice that we have a case of *leakage* whenever an output takes place of a level *lower* than the level of one of the inputs. Therefore, a flow from level σ to level τ is authorised if and only if $\sigma \leq \tau$. In line with this intuition, subtyping for processes is covariant in its second argument and contra-variant in its third argument.

Definition 4.1 A *type environment* Γ is a mapping from channel names to security levels such that $\Gamma(a) = \Gamma(\overline{a})$: we write $a{:}\ell \in \Gamma$ whenever $\Gamma(a) = \ell$.

Security type assignment on processes is defined by the following natural deduction system;

$$(\textsc{Nil}) : \quad \frac{}{\Gamma \vdash \mathbf{0} : (\bot, \top)}$$

$$(\textsc{Input}) : \quad \frac{\Gamma \vdash P : (\sigma, \tau)}{\Gamma \vdash a_\rho(v).P : (\rho \sqcup \sigma, \tau)} \; (a{:}\rho \in \Gamma, \rho \leq \tau)$$

$$(\textsc{Output}) : \frac{\Gamma \vdash P : (\sigma, \tau)}{\Gamma \vdash \overline{a_\rho}\langle e \rangle.P : (\sigma, \rho \sqcap \tau)} \; (a{:}\rho \in \Gamma, \sigma \leq \rho)$$

$$(\textsc{Par}) : \frac{\Gamma \vdash P : (\sigma_1, \tau_1) \quad \Gamma \vdash Q : (\sigma_2, \tau_2)}{\Gamma \vdash P | Q : (\sigma_1 \sqcup \sigma_2, \tau_1 \sqcap \tau_2)} \; (\sigma_1 \leq \tau_2 \;\&\; \sigma_2 \leq \tau_1)$$

$$(\textsc{Sum}) : \frac{\Gamma \vdash a_i.P_i : (\sigma, \tau)}{\Gamma \vdash \sum_{i \in I} \alpha_i.P_i : (\sigma, \tau)}$$

$$(\textsc{Restr}) : \frac{\Gamma, a{:}\rho \vdash P : (\sigma, \tau)}{\Gamma \vdash (\nu a_\rho)\, P : (\sigma, \tau)}$$

$$(\textsc{Rec}) : \frac{\Gamma, x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n \vdash P : (\sigma, \tau)}{\Gamma, a_1{:}\sigma_1, \ldots, a_n{:}\sigma_n \vdash A[\vec{a}] : (\sigma, \tau)}\ (A[\vec{x}] \triangleq P)$$

$$(\textsc{SubType}) : \frac{\Gamma \vdash P : (\sigma_1, \tau_1)}{\Gamma \vdash P : (\sigma_2, \tau_2)}\ (\sigma_1 \leq \sigma_2 \leq \tau_2 \leq \tau_1)$$

The side-conditions on levels guarantee than the input level never becomes bigger that the output level. For instance, a program of type $(\bot, \top)$ is guaranteed to not perform any input on a high channel nor any output on a low channel.

Our type system aims to capture the property that in the presence of an output, which is the means for an observer to deduce implicit flows in the program, any previous input has to be done at a lower level. Thus, a secure programs is one that for instance never emits an output. A secure program is also one that after every input emits only output of higher level, as expressed by the type (σ, τ) where $\sigma \leq \tau$. These property are preserved by subject reduction as shown.

Proposition 4.2 (Subject Reduction) *If* $\Gamma \vdash P : (\sigma, \tau)$ *and* $P \xrightarrow{\alpha} P'$ *then* $\Gamma \vdash P' : (\sigma, \tau)$.

Proof. By induction on $\Gamma \vdash P : (\sigma, \tau)$. □

We report in this section some examples of processes to show the power of discrimination of our type system. Some examples are taken from [5].

Example 4.3 Consider $a_h(r).\overline{b_l}\langle r\rangle$, $\overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle$, and $a_h(v).b_l(r)$. None of these processes is considered secure under BNDC.

$$(\nu\mathcal{H})(\overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle \mid \Pi) \not\approx (\nu\mathcal{H})(\overline{a_h}\langle r\rangle.\overline{b_l}\langle r\rangle)$$

This process is not secure because a high level action, either input or output, precedes a low level action. Our type system distinguishes between either high-level input or high-level output performed before a low-level action. We first consider $\overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle.\mathbf{0}$.

$$\cfrac{\cfrac{\Gamma, a{:}h, b{:}l \vdash \mathbf{0} : (l, h)}{\Gamma, a{:}h, b{:}l \vdash \overline{b_l}\langle r\rangle.\mathbf{0} : (l, l)}}{\Gamma, a{:}h, b{:}l \vdash \overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle.\mathbf{0} : (l, l)}$$

We now consider $a_h(v).b_l(r).\mathbf{0}$.

$$\dfrac{\dfrac{\Gamma, a{:}h, b{:}l \vdash \mathbf{0} : (l,h)}{\Gamma, a{:}h, b{:}l \vdash \overline{b_l}\langle r\rangle.\mathbf{0} : (l,l)}}{\Gamma, a{:}h, b{:}l \vdash a_h(v).\overline{b_l}\langle r\rangle.\mathbf{0} : (?,?)}\,(h \not\leq l)$$

In our type system, this process is not secure.

Example 4.4 We consider now the process $\overline{a_h}\langle v\rangle.\overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle$ is secure in BNDC but not in P-BNDC:

$$\dfrac{\dfrac{\Gamma, a{:}h, b{:}l \vdash \overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle : (l,l)}{\Gamma, a{:}h, b{:}l \vdash \overline{a_h}\langle v\rangle.\overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle : (l,l)} \quad \Gamma, a{:}h, b{:}l \vdash \overline{b}\langle r\rangle.\mathbf{0} : (l,l)}{\Gamma, a{:}h, b{:}l \vdash \overline{a_h}\langle v\rangle.\overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle : (l,l)}$$

The processes $a_h(v).a_h(v).\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle$ would be still secure in BNDC but not in P-BNDC, while clearly this process is not typeable in our type system.

Example 4.5 The process $a_l(x) + \overline{b_h}\langle e\rangle$ which is not included in neither the BNDC nor in the P-BNDC is secure according to the current type system.

$$\dfrac{\Gamma, a{:}l, b{:}h \vdash a_l(x) : (l,l) \quad \Gamma, a{:}l, b{:}h \vdash \overline{b_h}\langle r\rangle : (l,l)}{\Gamma, a{:}l, b{:}h \vdash a_l(x) + \overline{b_h}\langle e\rangle : (l,l)}$$

Clearly by the examples presented above, if $\Gamma \vdash P : (\sigma, \tau)$ then $P \notin$ BNDC nor $P \notin$ P-BNDC. It remains an interesting question what equivalence relation could be characterised by this type system.

We propose here a candidate which is a variation on both the BNDC and P-BNDC and we leave for future work to analyse the formal relationship with the type system.

Definition 4.6 [Refined Weak Bisimulation up-to-high] A symmetric binary relation $\mathcal{S} \subseteq \mathcal{P}r \times \mathcal{P}r$ is a *refined weak bisimulation up-to-high* if and only if $P\mathcal{S}Q$ implies that, for all $a \in Act$ either

- if $P \xrightarrow{\alpha} P'$ and $\alpha = \overline{a}e$ then there exists Q' such that $Q \overset{\widetilde{\alpha}}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$; or
- if $P \xrightarrow{\tau} P'$ then there exists Q' and a channel name a_ρ and a value e such that $\alpha = \overline{a}e$ and $Q \overset{\widetilde{\alpha}}{\Rightarrow} Q'$ and $subj(\alpha) = a_\rho, \rho = h$ and $P'\mathcal{S}Q'$ or $Q \overset{\widetilde{\tau}}{\Rightarrow} Q'$ and and $P'\mathcal{S}Q'$; or
- f $P \xrightarrow{\alpha} P'$ and $\alpha = ae$ then there exists Q' such that $Q \overset{\alpha}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$.

Two processes P, Q are *refined weakly bisimilar up-to-high*, written $P \approx^{I}_{\mathcal{H}} Q$, if $P\mathcal{S}Q$ for some refined weak up-to-high bisimulation $\mathcal{S}$.

The definition of Refined Weak bi-simulation up-to-high aims to distinguish between inputs and outputs. It is designed with the principles described below.

- High-level outputs can be matched by weak transition of the same name or any sequence of τ-actions.
- τ-actions can be matched either by any sequence of τ-actions. or by weak transitions of high-level output.
- Input can be matched only by weak transitions of the same name regardless the security level.

Definition 4.7 Let $A \subset \mathcal{H}$. We define $\Phi_A = \prod_{a \in A} a_h(z) \mid \Phi_A$.

Definition 4.8 [W-BNDC] P is said to be *generally secure*, $P \in$ W-BNDC if $(\nu\mathcal{H})(P \mid \Phi_{fn(P)}) \approx^I_{\mathcal{H}} P$.

The process Φ_A generate high-level input of the channels contained in A.

According to this definition of the process $a_h(v).a_h(v).\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle$ would not be considered generally secure. In fact, $(\nu\mathcal{H})(a_h(v).a_h(v).\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle \mid \Phi_a)$ the left-hand of the sum is blocked. Let's consider:

$$E = (\nu\mathcal{H})(a_h(v).a_h(v).\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle \mid \Pi) \qquad G = (a_h(v).a_h(v).\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle$$

Assume that $G \xrightarrow{ae} G'$ then E can only stay put. It is not difficult to show that $G \approx^I_{\mathcal{H}} E$ does not hold. If $E \xrightarrow{br} \mathbf{0}$ then G cannot match it with any low-level action.

On the other hand $\overline{a_h}\langle v\rangle.\overline{a_h}\langle v\rangle.\overline{b_l}\langle r\rangle + \overline{b_l}\langle r\rangle$ is generally secure.

Conclusions

In this paper we have considered two different approaches to non-interference, namely a static approach via a simple type system and a semantic approach via P-BNDC. We have shown that the 'simple type system' is correct with respect to P-BNDC, yet not complete. We have also defined a new type system that distinguishes between information flows from inputs and outputs. Information flow from high -level outputs to low-level channels is considered safe in the new type system. We defined also the Refined Weak Bi-simulation up-to-high which aims to characterise the refined type system.

As far as future work is concerned it would be interesting to relate typed language-based notion of non-interference with a process algebraic approach similarly to the work done in [5] for typed languages. In particular, it would be interesting to consider the type system of Volpano [14] or Boudol and Castellani [1] to define a type system in the process language that preserves that notion of non-interference.

Acknowledgements

We gratefully acknowledge Ilaria Castellani for many useful discussions and for having pointed out mistakes in the proof of Theorem 3.4 and for suggesting corrections. She also observed that the P-BNDC is not closed under general contexts and provided the example reported in this paper. We gratefully acknowledge the Group MIMOSA at INRIA Sophia-Antipolis where this work was initially conceived for their hospitality during 2005-2006.

References

[1] G. Boudol and I. Castellani. Non-interference for Concurrent Programs and Thread Systems, *Theoretical Computer Science* 281(1): 109-130, 2002.

[2] S. Crafa and S. Rossi. A Theory of Non-interference for the π-calculus. In *Proceedings of Symposium on Trustworthy Global Computing '05*, volume 3705 of *LNCS*, Springer-Verlag, 2005.

[3] R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In *Foundations of Security Analysis and Design - Tutorial Lectures* (R. Focardi and R. Gorrieri, Eds.), volume 2171 of *LNCS*, Springer, 2001.

[4] R. Focardi and S. Rossi Information Flow Security in Dynamic Contexts In *Proceedings of of the IEEE Computer Security Foundations Workshop*, pages 307–319, EEE Computer Society Press, 2002.

[5] R. Focardi and S. Rossi and A. Sabelfeld. Bridging Language-Based and Process Calculi Security. In *Proceedings of FoSSaCs'05*, volume 3441 of *LNCS*, Springer-Verlag, 2005.

[6] M. Hennessy and J. Riely. Information flow vs resource access in the asynchronous π-calculus. *ACM TOPLAS* 24(5): 566-591, 2002.

[7] M. Hennessy. The security π-calculus and non-interference. *Journal of Logic and Algebraic Programming* 63(1): 3-34, 2004.

[8] K. Honda and V. Vasconcelos and N. Yoshida. Secure information flow as typed process behavior. In *Proceedings of ESOP'00*, volume 1782 of *LNCS*, pages 180-199. Springer-Verlag, 2000.

[9] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Proceedings of P : (OP : (L'02*, pages 81-92. January, 2002.

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

[11] F. Pottier, A Simple View of Type-Secure Information Flow in the π-Calculus. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 320–330, 2002.

[12] A. Sabelfeld and D. Sands. Probabilistic Non-interference for Multi-threaded Programs. In *Proceedings of 13th Computer Security Foundations Workshop*, IEEE, 2000.

[13] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[14] D. Volpano and G. Smith and C. Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security* 4(3):167–187, 1996.