A Simple View of Type-Secure Information Flow in the $\pi$-Calculus

François Pottier
INRIA
E-mail: Francois.Pottier@inria.fr

Abstract

One way of enforcing an information flow control policy is to use a static type system capable of guaranteeing a noninterference property. Noninterference requires that two processes with distinct “high”-level components, but common “low”-level structure, cannot be distinguished by “low”-level observers. We state this property in terms of a rather strict notion of process equivalence, namely weak barbed reduction congruence.

Because noninterference is not a safety property, it is often regarded as more difficult to establish than a conventional type safety result. This paper aims to provide an elementary noninterference proof in the setting of the $\pi$-calculus. This is done by reducing the problem to subject reduction – a safety property – for a nonstandard, but fairly natural, extension of the $\pi$-calculus, baptized the $\langle\pi\rangle$-calculus.

1 Introduction

Information flow analysis consists in analyzing a program so as to determine how its output depends on the inputs it is given; or, more generally, how its observable behavior depends on the stimuli provided by its environment. Such an analysis allows static enforcement of so-called "information flow control" security policies, which prevent secret data from being leaked on public communication channels, or unreliable information from affecting critical decisions. More generally, dependency analysis is at the heart of several program transformation techniques [1].

Information flow analysis, reformulated as a type inference problem, has been heavily studied in the past few years, especially in the area of high-level, sequential languages; for references, see e.g. [20]. Recently, researchers [9, 11, 22, 10, 12, 27] have begun further extending this study to low-level, concurrent calculi, such as the $\pi$-calculus. This is made clearly worthwhile by the fact that these calculi allow modeling distributed computing systems and protocols, which today are at the heart of many real-world security concerns.

An information flow analysis usually comes in the form of a type system, together with a soundness proof. The former is typically (although not necessarily) derived from an existing system, whose types are augmented with annotations taken from a fixed security lattice $\mathcal{L}$ [3]. The soundness theorem gives a noninterference statement: it asserts that, if an appropriate type judgement holds, then no change in the high-security inputs of a process can affect a low-security observer. (Here, the usual subject reduction theorem becomes of rather secondary importance; nevertheless, it is often useful as a tool in the noninterference proof.) Such a statement must rely on some (preferably standard) notion of process equivalence, giving rise to a variety of choices.

Examining the theoretical results offered by existing works, we find some of them to be somewhat lacking in strength or simplicity. Hepburn and Wright [10] propose a type system which enjoys subject reduction, but do not establish any kind of security property. Hennessy and Riely [9] and Hennessy [8] base their noninterference statements on may- or must-testing equivalence; a statement based on a stronger notion of process equivalence would seem preferable. Sewell and Vitek [22] do not prove noninterference. They instead define a so-called “causal flow property” concerning the traces of a process in a “colored” labelled transition semantics. Because both the property and the colored semantics are ad hoc, it is difficult to determine exactly what is being guaranteed. Honda, Yoshida, Vasconcelos and Berger, in a number of papers [11, 12, 27] propose several advanced type systems, which allow exploiting linearity and deadlock-freedom properties to refine the information flow analysis. Their noninterference result, which is stated in terms of a weak bisimulation, requires a rather involved proof [26], due to the need to keep track of liveness properties. Furthermore, the type systems presented in [12, 27] have a rigid type structure (due to so-called “IO alternation” and “sequentiality” constraints), which is meant to allow encoding sequential computation only. They would need to be extended if they were to ac-
cept every $\pi$-calculus process that is well-typed in a simple type discipline.

In this paper, we wish to define a simple type-based information flow analysis, roughly equivalent in expressive power to Hennessy and Riely’s [9], to Sewell and Vittek’s [22], or to the nonlinear fragment of Honda et al.’s [11], and to give an elementary noninterference proof for it. By showing how straightforward it is to establish noninterference in this simple setting, we hope to convey some useful insights, and to later facilitate the understanding of more advanced type systems.

Our noninterference result will be stated in terms of bisimulation equivalence, rather than (say) may-testing equivalence, so as to identify more processes as insecure. (See Focardi and Gorrieri’s work [7] for a discussion of this issue.) In short, relying on a bisimulation equivalence allows detecting information leaks caused by contention between “low”-level and “high”-level processes waiting on a single channel $x$. This, we argue, sounds desirable, because, in practice (that is, under a reasonable fairness assumption), a “low”-level process may detect the presence of a “high”-level one by noticing that a message that was sent to it on channel $x$ was not received after a certain amount of time.

Of course, one may object that this phenomenon is only a particular kind of timing leak. Our type system does not, in general, detect such leaks. Neither does it detect probabilistic information leaks, because we use notions of process equivalence based on possible, rather than likely, behaviors. Still, we argue that weak bisimulation yields a stronger noninterference result than may-testing equivalence, and we choose to rely on it.

## 2 Overview

We wish to establish a noninterference result, i.e. prove that two (well-typed) processes which differ only in some “high”-level components behave identically when observed through “low”-level channels. Note that this is not a so-called safety property, because it requires examining the traces of two processes, rather than of a single one. (See e.g. [15] for more details.) This explains why noninterference is considered more difficult to establish than a conventional type safety result. Our approach consists in reducing noninterference to a simple subject reduction property (i.e. to a safety property) for a nonstandard extension of the $\pi$-calculus. It is inspired by previous joint work with Vincent Simonet [21].

Our calculus, baptized the $\langle \pi \rangle$-calculus, describes the independent execution of a pair of (well-typed) processes which we arbitrarily index by $\{1, 2\}$, while keeping track of their shared sub-processes. For instance, the $\langle \pi \rangle$-calculus process $P \mid (Q)_1 \mid (R)_2$ represents the pair $(P \mid Q, P \mid R)$, while explicitly recording the fact that the sub-process $P$ is shared. A sub-term of the form $(P)_i$ indicates that $P$ is present only in the process of index $i$, and absent in the one of index $j$, where $\{i, j\} = \{1, 2\}$. Brackets cannot be nested. This would not make any sense, since we only wish to encode two standard processes, not more.

Because we are interested in comparing two processes that differ only in their “high”-level components, and because we encode these differences using “bracket” constructors, any sub-process that appears within brackets will be considered a “high”-level process. Conversely, processes that are shared will be considered “low”-level.

One may wonder why it is not sufficient to employ only one kind of brackets, i.e. a single construction $\langle \cdot \rangle$. For instance, the process $\bar{x} \mid \langle x. P \rangle$ would represent the pair of processes $(\bar{x}, \bar{x} \mid x. P)$, while encoding the fact that $x. P$ is considered a “high”-level process. However, such a language would not be rich enough to express reduction: indeed, the pair of processes above evolves to $(\bar{x}, P)$, which cannot be represented in the single-bracket syntax – hence the need for two distinct bracket constructs.

In Sect. 3, we equip the $\langle \pi \rangle$-calculus with an operational semantics which reflects that of the standard $\pi$-calculus, but preserves sharing information. Reduction may take place outside brackets (meaning that a common reduction step is performed), inside brackets (meaning that some process performs an independent, “high”-level step), or at bracket boundaries. In the latter case, we discard sharing information by reducing $P$ to $(P)_1 \mid (P)_2$. However, this is allowed only if some communication step cannot otherwise take place; gratuitous loss of sharing is forbidden. (This is a crucial point; because we consider any process that appears under brackets as “high”-level, the semantics should not cause brackets to appear needlessly, lest the precision of the information flow analysis be compromised.) We begin our formal development by showing how the $\langle \pi \rangle$-calculus relates to the standard $\pi$-calculus.

In Sect. 4, we equip the $\langle \pi \rangle$-calculus with a type system. It is very similar to Pierce and Sangiorgi’s type system for the $\pi$-calculus [18], but is extended with security annotations, a standard notion in information flow analyses; see e.g. [11, 22]. We prove that it enjoys a subject reduction property.

Sect. 5 shows how these results combine to yield a weak-bisimulation-based noninterference property. In short, the bisimulation diagram naturally arises by design of the $\langle \pi \rangle$-calculus, while preservation of “low”-level barbs is a simple consequence of the typing hypothesis. Then, for convenience, Sect. 6 rephrases our results using “colored” (rather than “bracket”) notation for processes.
3 The \(\langle \pi \rangle\)-Calculus

3.1 Presentation

The \(\langle \pi \rangle\)-calculus is an extension of the synchronous polyadic \(\pi\)-calculus [16], whose semantics describes the independent execution of a pair of processes, while keeping track of their shared sub-processes. (We consider a synchronous variant, because it contains the asynchronous variant as a fragment; see Sect. 7.1 for comments about the asynchronous case.)

Definition 1 Let \(x, y, z, \ldots\) range over a denumerable set \(N\) of names. Let \(\hat{y}, \hat{z}, \ldots\) denote vectors of such names. Let \(i\) range over \(\{1, 2\}\). Normal processes \(M, N, \ldots\) and processes \(P, Q, R, \ldots\) are given by

\[
N ::= x(\hat{y}).P \mid z(\hat{z}).P \mid 0 \mid N + N
\]

\[
P ::= N \mid (P \mid P) \mid !P \mid \nu x.P \mid \langle P \rangle_i
\]

\(x(\hat{y}).P\) binds \(\hat{y}\) in \(P\), and \(\nu x.P\) binds \(x\) in \(P\). From this information, the usual notions of free names, capture-free substitution, and \(\alpha\)-convertibility are deduced. We write \(\text{fn}(P)\) for the set of free names of a process \(P\). We identify processes up to \(\alpha\)-conversion.

In the following, bold meta-variables denote standard processes, i.e., processes which have no sub-term of the form \(\langle P \rangle_i\). Throughout the paper, we restrict our attention to processes where every sub-term of the form \(\langle P \rangle_i\) is in fact of the form \(\langle P \rangle_i\), i.e., we never nest brackets.

A single \(\langle \pi \rangle\)-calculus process is meant to represent a pair of standard \(\pi\)-calculus processes. In particular, \(\langle P \rangle_1\) stands for the pair \((P, 0)\), while \(\langle P \rangle_2\) stands for \((0, P)\). A \(\langle \pi \rangle\)-calculus process of the form \(P\) represents shared structure: it stands for the pair \((P, P)\). More generally, an arbitrary process \(P\) stands for the pair \((\pi_1(P), \pi_2(P))\), where the projection functions \(\pi_1\) and \(\pi_2\) are defined as follows.

Definition 2 Let \(\{i, j\} = \{1, 2\}\). The \(i\)th projection function, written \(\pi_i\), satisfies the laws \(\pi_i(\langle P \rangle_j) = P\) and \(\pi_i(\langle P \rangle_j) = 0\) and is a homomorphism on other (i.e., standard) process forms. We often write \(\pi_iP\) for \(\pi_i(\langle P \rangle)\). For all \(P, \pi_iP\) is a standard process.

Projection may create superfluous null processes, which it is convenient to disregard. We introduce an auxiliary relation for this purpose.

Definition 3 Let \(\leq_0\) be the smallest reflexive, compatible relation over processes which satisfies the law \(P \leq_0 P \mid 0\).

(By compatible, we mean closed under all contexts.) Viewing \(\pi_i\) as a relation, we will write \(P \pi_i \cdot \geq_0 P\) or \(P \leq_0 \cdot \pi_i^{-1} P\) to denote \(\pi_iP \geq_0 P\). (The superscript \(*\) denotes the reflexive, transitive closure of a relation; the infix operator \(\cdot\) denotes the composition of relations.)

3.2 Semantics

We now define an operational semantics for the \(\langle \pi \rangle\)-calculus. Its restriction to standard processes will coincide with the standard synchronous polyadic \(\pi\)-calculus [16].

For purely technical reasons, we choose a slightly altered presentation of the latter as a starting point: we turn three structural congruence laws, namely scope extrusion, replication, and spontaneous creation of new bound names, into reduction rules, thus making them irreversible. Proving that these changes do not affect the semantics of the \(\pi\)-calculus is an orthogonal issue, which we do not address here.

For technical convenience, structural congruence is not made transitive. Similarly, evaluation contexts are not allowed to be nested. These presentational choices eliminate some redundancy from our definitions, thus simplifying proofs.

Definition 4 Structural congruence \(\equiv\) is the smallest reflexive, symmetric, compatible relation over processes such that the following laws hold:

1. \(N + 0 \equiv N, N_1 + N_2 \equiv N_2 + N_1, (N_1 + N_2) + N_3 \equiv N_1 + (N_2 + N_3)\);
2. \(P \mid 0 \equiv P, P_1 \mid P_2 \equiv P_2 \mid P_1, (P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)\);
3. \(\nu x.\nu y.P \equiv \nu y.\nu x.P\).

Definition 5 An evaluation context \(E\) is one of \(([] \mid P)\), \(\nu x.[] \) or \((\langle \rangle)\).

All structural congruence laws are standard. All evaluation contexts are standard as well, except \((\langle \rangle)\), which allows reduction under brackets.

Definition 6 The raw one-step reduction relation \(\rightarrow\) is given by Fig. 1. We write \(M \# N\) (read: \(M\) and \(N\) may communicate) for \((M \mid N) \rightarrow \vee (N \mid M) \rightarrow\), where \((P \rightarrow)\) is itself a short-hand for \((\exists P' \ P \rightarrow P')\). Weak reduction, written \(\Rightarrow\), is defined as \((\equiv \cup \rightarrow)^*\).

Rules COMM and CONTEXT are standard. Rules EXTR, REPL and NEW are directed versions of standard congruence rules. The crucial rule is SPLIT, which allows discarding sharing information if required by further reductions. SPLIT can be viewed as a restriction of the following, more liberal rule:

\[
\text{SPLIT'}
\]

\[
M \rightarrow \langle \pi_1M \rangle_1 \mid \langle \pi_2M \rangle_2
\]

SPLIT’ explicitly replaces a shared process with its projections. Thus, it implements our intuition that a \(\langle \pi \rangle\)-calculus process stands for a pair of standard processes. However,
because it has no side-condition, it allows sharing information to be discarded at will. Although such a behavior would be perfectly valid as far as the untyped semantics of the \(\langle\pi\rangle\)-calculus is concerned, it would lead to a useless type system. Indeed, as we have said, every process which appears under brackets must remain invisible to “low”-level observers. However, \textsc{Split*} potentially causes every process to appear under brackets. Because the type system must have subject reduction, adopting this rule would force it to typecheck every process under the most restrictive security assumption.

As a result, we must replace \textsc{Split*} with a restricted version that preserves as much sharing information as possible, i.e. that allows it to be discarded only if some communication step cannot otherwise take place. Thus, we obtain \textsc{Split}, where \(M\) can be split into \(\langle\pi_i M\rangle_i \mid \langle\pi_j M\rangle_j\) only if one of its projections, say \(\pi_i M\), is able to communicate with some term \(N\) which already appears under a bracket \(\langle N\rangle_i\).

Rules \textsc{Glue} and \textsc{Break} can be understood as restrictions of a (hypothetical) structural congruence rule:

\[
\langle P \mid Q\rangle_i \equiv \langle P' \rangle_i \mid \langle Q\rangle;
\]

Again, allowing this equivalence to hold would be correct as far as the untyped semantics is concerned, but would pose a slight technical problem. The premise in rule \textsc{Glue} works around it by allowing brackets to be merged only if required to allow some communication step. (Have a look at the subject reduction proof for \textsc{Glue} for more details.) Lastly, rule \textsc{Push} allows pushing brackets down inside \(\nu\) binders. In conjunction with \textsc{Break}, this lets them move down to the level of normal processes, where they can be dealt with by \textsc{Split}.

### 3.3 Relating the \(\langle\pi\rangle\)-Calculus to the \(\pi\)-Calculus

We now relate the \(\langle\pi\rangle\)-calculus to the \(\pi\)-calculus. (Because the latter is a fragment of the former, we do not define it separately.) Our aim is to show that the \(\langle\pi\rangle\)-calculus allows reasoning about the execution of a pair of standard \(\pi\)-calculus processes. That is, every reduction of a \(\langle\pi\rangle\)-calculus process represents correct reductions of its projections; conversely, every reduction of a projection can be emulated by reductions of the whole. In other words, the two semantics are in a weak bisimulation relation.

We begin with the easier part, i.e. proving that the projection of every reduction step is a correct reduction step as well. This is immediate; we omit the proof. (All missing proofs can be found in the full version of this paper [19].)

**Lemma 1** Let \(i \in \{1, 2\}\). If \(P \Rightarrow P'\), then \(\pi_i P \Rightarrow \pi_i P'\).

We continue with the subtler part, i.e. proving that every correct (congruence or reduction) step performed by a projection \(\pi_i P\) can be emulated by the term \(P\). There is a slight technical twist: if \(\pi_i P\) is \(P \mid 0\), which reduces to \(P\) via a structural congruence step, then \(P\) may be of the form \(P \mid \langle Q\rangle_j\), where \(\{i, j\} = \{1, 2\}\). In that case, \(P\) will be unable to perform the same step. To account for this, we use the pre-order \(\leq_{\pi}\) introduced in Definition 3.

**Lemma 2** Let \(i \in \{1, 2\}\). If \(P \Rightarrow P'\) and \(P \leq_{\pi} \pi_i^{-1} P\), then there exists some process \(P''\) such that \(P \Rightarrow P''\) and \(P'' \leq_{\pi}\pi_i^{-1} P''\).

The proof, which requires a number of auxiliary lemmas, is given in the full version of this paper [19].

### 4 Typing the \(\langle\pi\rangle\)-Calculus

#### 4.1 Presentation

We now introduce a type system for the \(\langle\pi\rangle\)-calculus. It extends an existing typing system for the \(\pi\)-calculus – namely Pierce and Sangiorgi’s [18], which we choose for its simplicity – with security annotations. Its typing judgements are of the form \(\Gamma \vdash_{\text{pc}} P\), where pc is a security level, i.e. a member of a fixed security lattice \(\mathcal{L}\). Such a judgement may be read: under assumptions \(\Gamma\), \(P\) is well-typed and will affect only observers of security clearance pc or higher. It may also be read: under assumptions \(\Gamma\), assuming \(P\) gains
information of level pc by being executed. P is well-typed.
This formulation explains why this meta-variable is historically named pc [6]: it is the security level which P attains simply by virtue of being executed, i.e. the security level associated with its “program counter”.

Even though the security lattice \( \mathcal{L} \) is arbitrary, it is desirable to establish a simple dichotomy between “low” and “high” security levels. Such a distinction allows simple proofs; full generality will be recovered in Sect. 6. To this end, in the present section, we assume given a fixed, downward-closed set \( L \subseteq \mathcal{L} \). We will view levels within (resp. outside) \( L \) as “low” (resp. “high”).

Noninterference states that two processes which differ only in some high-level sub-terms cannot be distinguished both. It is further annotated with a security level associated with its “program counter”.

Definition 7 A polarity \( p \) is one of \( \{ - , + , \pm \} \). Types \( t \) are of the form \( \langle \hat{t} \rangle^p \).

Definition 8 Polarieties are ordered by \( \pm \leq - , + \leq \pm \). Types are ordered by

\[
\langle \hat{i} \rangle^p \leq \langle \hat{i}' \rangle^{p'} \quad (p' \leq - \Rightarrow \hat{i} \leq \hat{i}') \quad (p' \leq + \Rightarrow \hat{i} \leq \hat{i}')
\]

(The ordering is extended point-wise to vectors of types.)

These definitions can be understood either inductively or co-inductively, yielding finite or infinite types. The choice of one or the other is orthogonal to our concerns; indeed, the subject reduction and noninterference proofs are entirely independent of this issue. We leave it open, to be settled by the analysis designer at a later stage.

Note that our definition of subtyping does not allow a channel’s security level to be modified, be it covariantly or contravariantly. In other words, two channel types that are in a subtyping relationship must have the same security level. This (admittedly strong) requirement reflects the fact that information flows both ways along a channel, regardless of the direction of messages (i.e. regardless of the channel’s polarity). This property will be used in the proof of Lemma 6, which itself plays a key role in establishing subject reduction.

---

**Figure 2. Typing rules of the \( \langle \pi \rangle \)-calculus**

**Definition 9** The type system of the \( \langle \pi \rangle \)-calculus is defined by Fig. 2. It involves two separate judgement forms. Judgements of the form \( \Gamma \vdash_{pc} N \) concern normal processes. Judgements of the form \( \Gamma \vdash_{pc} P \) concern arbitrary processes.

Rules T-RECV and T-SEND require the channel’s security level to match the level attained by the process, namely pc. Furthermore, T-SUM requires all components of a sum to have matching levels. As a result, if \( \Gamma \vdash_{pc} N \) holds, then all channels liable to be read or written by \( N \) must have the same security level. This reflects the fact that information may flow arbitrarily between these channels. As a simple illustration of this fact, consider \( N = x.P + y.Q \). If, in the presence of \( N \), a message sent on channel \( x \) is not consumed after a while, then its sender knows that some message was available on channel \( y \), so information flows from \( y \) to \( x \). (Recall that relying on weak bisimulation amounts, in practice, to formulating a fairness hypothesis.) By symmetry, the converse is also true.

Rule T-SUB allows strengthening the security requirements bearing on a process. This rule applies only to judgements of the form \( \Gamma \vdash_{pc} P \); applying it to judgements of the form \( \Gamma \vdash_{pc} N \) would break the property that all components of a sum have a common security level.

In T-RECV and T-SEND, the continuation process \( P \) is typed at a security level equal to that of the channel \( x \) (or greater, thanks to T-SUB). This reflects the fact that a successful synchronization at \( x \) yields information which \( P \)
may exploit.

4.2 Type Preservation

We begin with three easy lemmas, stating that typing is preserved by projection, by structural congruence, and by substitution of names for names. Proofs are omitted.

Lemma 3 For \( i \in \{1, 2 \} \), \( \Gamma \vdash_{pc} P \) implies \( \Gamma \vdash_{pc} \pi_i P \).

Lemma 4 \( \Gamma \vdash_{pc} P \) and \( P \equiv P' \) imply \( \Gamma \vdash_{pc} P' \).

Lemma 5 \( \Gamma; \bar{\gamma}: i \vdash_{pc} P \) and \( \Gamma(\bar{z}) \leq i \) imply \( \Gamma \vdash_{pc} P[\bar{z}/\bar{\gamma}] \).

Next comes an important auxiliary lemma, stating that, if two normal processes \( M \) and \( N \) are able to communicate with each other, and if they are typed under a common environment \( \Gamma \), then they must be typed at the same security level. (Additionally, the lemma states that the reduced process \( R \) is well-typed at that level.) Indeed, \( M \) and \( N \) must share some channel \( x \), so they must both be typed at \( x \)'s security level.

Lemma 6 Assume \( \Gamma \vdash_{(pc_1)} M \) and \( \Gamma \vdash_{(pc_2)} N \). If \( M \mid N \rightarrow R \), then \( \Gamma \vdash_{pc} R \) holds, where \( pc = pc_1 = pc_2 \).

Proof. Because \( M \) and \( N \) are normal processes, they are irreducible; thus, the reduction \( M \mid N \rightarrow R \) must be an instance of rule COMM. Thus, \( M \) and \( N \) must be of the form \( M' + x(\bar{y}) \) and \( N' + x(\bar{z}) \), respectively.

By T-SUM and T-RECV, the first hypothesis yields \( \Gamma(\bar{y}) \leq \langle t_1 \rangle_{pc_1} \), and \( \Gamma; \bar{\gamma}: \bar{y} : t_1 \vdash_{pc_1} P \). By T-SUM and T-SEND, the second one yields \( \Gamma(\bar{z}) \leq \langle t_2 \rangle_{pc_2} \), \( \Gamma(\bar{z}) \leq t_2 \) and \( \Gamma \vdash_{pc_2} Q \).

From \( \Gamma(\bar{z}) \leq \langle t_2 \rangle_{pc_2} \), \( \Gamma(\bar{z}) \leq t_2 \) and \( \Gamma \vdash_{pc_2} Q \), we deduce \( pc_1 = pc_2 \) and \( t_2 \leq t_1 \). Take \( pc = pc_1 = pc_2 \). By transitivity of subtyping, \( \Gamma(\bar{z}) \leq t_1 \) holds. Lemma 5 then yields \( \Gamma \vdash_{pc} P[\bar{x}/\bar{y}] \). Using \( \Gamma \vdash_{pc} P[\bar{x}/\bar{y}] \mid Q \), that is, \( \Gamma \vdash_{pc} R \). \( \Box \)

These results allow us to establish that typing is preserved by reduction.

Lemma 7 (Subject Reduction) \( \Gamma \vdash_{pc} P \) and \( P \rightarrow P' \) imply \( \Gamma \vdash_{pc} P' \).

Proof. By induction on the derivation of \( P \rightarrow P' \).

Case PRED. \( P \) is of the form \( M \mid N \). By T-SUB, T-PAR and T-NORMAL, this yields \( \Gamma \vdash_{(pc_1)} M \) and \( \Gamma \vdash_{(pc_2)} N \), where \( pc \leq pc_1 \) and \( pc \leq pc_2 \) hold. By Lemma 6 and by T-SUB, \( \Gamma \vdash_{pc} P' \) holds.

Case SPLIT. The reduction is \( M \mid \langle N \rangle_i \rightarrow \langle \pi_i M \mid N \rangle_i \mid \langle \pi_j M \rangle_j \), where \( \{i, j\} = \{1, 2\} \) and \( \pi_i M \# N \). Our hypothesis is \( \Gamma \vdash_{pc} M \mid \langle N \rangle_i \). By T-SUB, T-PAR, T-NORMAL and T-BRACKET, this yields \( \Gamma \vdash_{(pc_1)} M \) and \( \Gamma \vdash_{pc_2} N \), where \( pc \leq pc_1 \), \( pc \leq pc_2 \), and \( pc_2 \not\in L \) hold. Because \( L \) is downward-closed, we may assume, without loss of generality, that T-SUB is never used immediately above T-BRACKET; as a result, \( \Gamma \vdash_{(pc_2)} N \) holds. \( \pi_i M \# N \) implies \( M \# N \). As a result, we may apply Lemma 6 (either to \( M \) and \( N \) or to \( M \) and \( N \)), yielding \( pc_1 = pc_2 \).

It follows that \( \Gamma \vdash_{pc_2} M \) holds. By Lemma 3, both \( \pi_i M \) and \( \pi_j M \) are well-typed under \( \Gamma \) and \( pc_2 \). By hypothesis, so is \( N \). Thus, considering \( pc_2 \not\in L \), so is \( \langle \pi_i M \mid N \rangle_i \mid \langle \pi_j M \rangle_j \). Because \( pc_2 \geq pc \), the result follows by T-SUB.

Case GLUE. The reduction is \( \Gamma \vdash_{pc} \langle M \rangle_i \mid \langle N \rangle_i \), where \( M \# N \). Our hypothesis is \( \Gamma \vdash_{pc} \langle M \rangle_i \mid \langle N \rangle_i \). As above, this yields \( \Gamma \vdash_{(pc_1)} M \) and \( \Gamma \vdash_{(pc_2)} N \), where \( pc \leq pc_k \) and \( pc_k \not\in L \) for \( k \in \{1, 2\} \). Again, Lemma 6 yields \( pc_1 = pc_2 \). Thus, \( \Gamma \vdash_{pc} \langle M \mid N \rangle \) holds. The result follows by T-SUB.

Case EXTR, REPL, CONTEXT, BREAK, PUSH. Immediate. \( \Box \)

5 Noninterference

Combining the results of Sect. 3.3 and 4.2, we will now derive a noninterference property, expressed in terms of weak barbed reduction congruence [17].

Definition 10 Let \( \alpha \) range over names and co-names \( (x, x, \ldots) \). If \( \alpha \) is \( x \) (resp. \( \bar{x} \)), then \( \alpha \) stands for \( x \) (resp. \( \bar{x} \)). If \( \alpha \) is \( x \) or \( \bar{x} \), then \( |\alpha| \) is \( x \). The predicate \( P \downarrow_\alpha \) (read: the process \( P \) is observable at \( \alpha \)) is defined as follows:

\[
(M + x(\bar{y})).P \downarrow_\alpha \quad \text{(read: observable \( P \) for } \alpha) \quad (M + \bar{x}(\bar{y})).P \downarrow_\alpha
\]

\[
P \downarrow_\alpha \quad E \text{ does not bind } |\alpha|
\]

\[
P \downarrow_\alpha \text{ stands for } (\exists P' \quad P \Rightarrow P' \wedge P' \downarrow_\alpha).
\]

Definition 11 Let \( B \) be an arbitrary set of names. A binary relation \( \mathcal{R} \) over processes is a weak \( B \)-simulation if and only if

- \( P \mathcal{R} Q \wedge P \Rightarrow P' \) implies \( \exists Q' \quad Q \Rightarrow Q' \wedge P' \mathcal{R} Q' \); and

- \( |\alpha| \in B, P \mathcal{R} Q \) and \( P \downarrow_\alpha \) imply \( Q \downarrow_\alpha \).

\( \mathcal{R} \) is a weak \( B \)-bismulation if and only if \( \mathcal{R} \) and \( \mathcal{R}^{-1} \) are weak \( B \)-simulations. Two processes are weakly \( B \)-bisimilar if they are related by some weak \( B \)-bismulation. They are weakly bisimilar if they are \( N \)-bisimilar.

In this section, \( L \) is fixed, as in Sect. 4. The set of channels which, according to a type environment \( \Gamma \), do not leak any high-level information is referred to as low(\( \Gamma \)). It is defined as follows.
Theorem 1

For any process $R$, for any (standard) context $C$ such that $\Gamma \vdash C[R]$ holds in some $L$-environment $\Gamma$, $C[\pi_1 R]$ and $C[\pi_2 R]$ are weakly bisimilar.

Proof. Apply Lemma 9. For $i \in \{1, 2\}$, $\pi_i C[R]$ is $C[\pi_i R]$. As a result, $C[\pi_1 R]$ and $C[\pi_2 R]$ are related by $R_{\Gamma}$. Thus, they are weakly low($\Gamma$)-bisimilar. The result follows from low($\Gamma$) = $\mathcal{N}$.

Our second corollary is in the style of Honda et al.’s non-interference claim [11]:

Theorem 2

Assume $\Gamma_0 \vdash_{pc} P_i$, where $pc_0 \not\in L$, holds for $i \in \{1, 2\}$. Then, for any context $C$ and for any environment $\Gamma$ such that $\Gamma \vdash C[i]$ holds under the assumption $\Gamma_0 \vdash_{pc} [], C[P_1]$ and $C[P_2]$ are weakly low($\Gamma$)-bisimilar.

Proof. Because $pc_0 \not\in L$, T-BRACKET yields $\Gamma_0 \vdash_{pc_0} (P_i)i$ for $i \in \{1, 2\}$. Define $R$ as $C[(P_1)i \mid (P_2)i]$. According to $T$-PAR and to our hypothesis about $C$, $\Gamma \vdash R$ holds. Apply Lemma 9. $C[P_1]$ and $C[P_2]$ are related by $R_{\Gamma}$. As a result, they are weakly low($\Gamma$)-bisimilar.

Theorem 1 is, in our opinion, more directly useful than Theorem 2. Indeed, the former allows precise reasoning about two processes which have some common structure, while the latter treats $P_1$ and $P_2$ as entirely separate. However, Theorem 1 is expressed in terms of a nonstandard theoretical tool, namely the $<\pi>$-calculus. For this reason, we will now reformulate it, with a practical aim: provide a specification of the security guarantees offered by the type system that can be read and written by the programmer.

6 Programmer-Oriented Specification

We suggest allowing the programmer to color any sub-process with a security level $l \in L$, indicating that its presence should not affect observers whose security clearance is not at least $l$. Thus, we introduce a “colored” $\pi$-calculus. It is reminiscent of Abadi, Lampson, and Lévy labelled $\lambda$-calculus [2] and of Sewell and Vitek’s colored box-$\pi$-calculus [22]. However, we do not need to define a semantics for it; here, we view it only as a programming notation.

$$P ::= [x \bar{\pi}].P \mid P \{x \mid P \} \mid 0 \mid N + N \mid P : l$$

The only variation with respect to the $<\pi>$-calculus is that the construct $(P)l$ is replaced with the coloring construct $l : P$. (Unlike brackets, coloring constructs can be nested.) The type system is modified by replacing rule T-BRACKET with

$$\frac{\begin{array}{c}
T-COLOR
\Gamma \vdash_{pc} P \mid l \leq pc
\hline
\Gamma \vdash_{pc} l : P
\end{array}}{}$$

Note that the new type system is no longer parameterized by a set $L$. We use $\Gamma \vdash_{pc} P$ to denote its judgements.
Rule T-COLOR simply forces the sub-process $P$ to be typechecked at a level that equals or exceeds $l$. Thus, “colors” in the source program cause the typechecker to enforce additional constraints, which will then guarantee a certain security property, as we will now explain.

Given $L \subseteq \mathcal{L}$, we define an erasure function $[\cdot]_L$, which drops all sub-terms whose color is not a member of $L$, and produces a term in the standard $\pi$-calculus. The function which strips off all colors, namely $[\cdot]$, is written $\cdot$.

Definition 13 Let $[\cdot]_L$ satisfy $[l : P]_L = [P]_L$ when $l \in L$, $[l : P]_L = 0$ when $l \not\in L$, and be a homomorphism on standard process forms.

Then, our final noninterference theorem states that pruning sub-terms which carry “high” colors does not alter the behavior of a process under a certain typed barbed congruence, whereby a context is allowable only if its type states that it will affect “low” channels only. Note that the meaning of “low” and “high” is parameterized by the choice of $L$, of which the type system is now independent.

Theorem 3 (Noninterference) Let $L$ be a downward-closed subset of $\mathcal{L}$. For any process $R$ of the colored $\pi$-calculus, for any (standard) context $C$ such that $\Gamma \vdash C[R]$ holds in some $L$-environment $\Gamma$, $C[[R]]$ and $C[[R]]_L$ are weakly bisimilar.

Proof. Define a mapping $\cdot$ from the colored $\pi$-calculus into the $\langle \pi \rangle$-calculus, which satisfies $[l : P]^* = P^*$ when $l \in L$, $[l : P]^* = ([P])^*_1$ when $l \not\in L$, and is a homomorphism on standard process forms. Then, the identities $\pi_1(P^*) = [P]$ and $\pi_2(P^*) = [P]_L$ hold for any colored process $P$. Furthermore, it is easy to check that $\Gamma \vdash_{pc} P$ implies $\Gamma \vdash P^*$. As a consequence, $\Gamma \vdash C[R]^*$ yields $\Gamma \vdash (C[R])^*$. Furthermore, $(C[R])^*$ is weakly bisimilar. By Theorem 1, $C[[\pi_1_R]]$ and $C[[\pi_2_R]]$ are weakly bisimilar. By the identities above, these are none other than $C[[R]]$ and $\Gamma \vdash C[[R]]_L$.

In particular, if $R$ itself is well-typed within some $L$-environment, then every context $C$ such that $C[[R]]$ is well-typed will do.

7 Discussion

7.1 Asynchrony

Our type system allows input (resp. output) channel types $\langle \cdot \rangle$- (resp. $\langle \cdot \rangle$-) to be covariant (resp. contravariant) in their parameters $\cdot$, but both are invariant in their security level $l$ (Definition 8). Intuitively, this is because synchronization causes information to flow not only from the sender of a message to its receiver, but also in the reverse direction. As a result, both processes must have the same notion of “pc”. This is enforced by the invariance of security annotations.

Let us now restrict our interest to the asynchronous fragment of the $\pi$-calculus, where every sender is of the form $P(z)$. It may seem that, under this restriction, senders can no longer observe the reception of their messages. This would suggest that making input (resp. output) channel types covariant (resp. contravariant) in their security level is safe. This, however, is not the case; in fact, message reception can still be observed by exploiting contention between receivers. This is illustrated by the following example. Take $L = \{l, h\}$, with $L \subseteq H$. Take $L = \{l\}$. Consider the typing judgement

$$x : \langle \cdot \rangle^+_l, y : \langle \cdot \rangle^+_l, z : \langle \cdot \rangle^+_l \vdash \tilde{\gamma} | x, y.0 | y, \tilde{\pi}$$

This judgement is incorrect in our type system, because the sub-term $x, y.0$ is ill-typed. Indeed, listening on channel $x$, which has type $\langle \cdot \rangle^+_l$, causes “pc” to become H. Subsequently, listening on channel $y$ becomes illegal, since $y$ has type $\langle \cdot \rangle^+_l$. On the other hand, if input channel types were covariant, then $\langle \cdot \rangle^+_l$ would be a subtype of $\langle \cdot \rangle^+_H$, and the judgement would become correct. Yet the two projections of this process, namely

$$\tilde{\gamma} | x, y.0 | y, \tilde{\pi} \quad \text{and} \quad \tilde{\pi} | x, y.0 | y, \tilde{\pi}$$

are not weakly $\langle y, z \rangle$-bisimilar. Indeed, the left-hand process must send a message on channel $z$, while the right-hand one may choose to never do so. In other words, the presence of $\tilde{\pi}$ causes contention between two receivers on $y$, which can be detected by observing $z$. This example shows that channel types must remain invariant in their security annotations, even in the asynchronous fragment of the $\pi$-calculus.

7.2 Exploiting Linearity

In the example above, the information leak is caused by contention. If the channel $y$ was linear [14], then no contention would be possible, and it would be safe for the “high”-level process $x, y.0$ to receive a signal through the “low”-level channel $y$. In fact, under a strong notion of linearity, every communication action on a linear channel must eventually succeed (see e.g. [25]), so its success alone does not carry any information. This fact is pointed out and exploited by Honda et al. [11, 12, 27]. It is indeed crucial, in practice, to take advantage of it, because the $\pi$-calculus is a low-level programming language, where continuation-passing style is ubiquitous: control is encoded through the use of (often linear) communications, rather than evident in the program’s syntax. Zdancewic and Myers [28] address this issue in the case of a low-level, sequential calculus.
Can our proof approach be extended to deal with linearity information? Let us give a rough sketch of how we envision such an extension. The semantics of the $\langle \pi \rangle$-calculus must be modified to disallow linear communications from taking place under brackets. Instead, one should introduce reduction rules akin to the following, which re-discovers sharing:

$$\text{JOIN}$$

$$\langle x.P \rangle_1 \mid \langle x.Q \rangle_2 \Rightarrow_\mu \langle x.P \rangle_1 \mid \langle x.Q \rangle_2$$

if $x$ is linear

These changes allow a continuation $x.P$ to be triggered without splitting $P$. In turn, this will allow the type system to view $P$ as a “low”-level process, even though the trigger $x$ is sent from a “high”-level process. The anticipated difficulty is in establishing the completeness lemma, i.e. the analogue of Lemma 2. Indeed, proving that JOIN is always applicable requires proving that every linear communication action will eventually succeed. This requires using typing information, whereas, in our current development, the results in Sect. 3.3 were (pleasantly) independent of types. We view this research direction as most promising.

7.3 Type Inference

Type inference is no more difficult for this system than for Pierce and Sangiorgi’s original type system [18]. Type inference for (an extended version of) the latter has been studied by Igarashi and Kobayashi [13]. To adapt their algorithm, one must generate and solve extra inequalities, as required by rules T-SUB and T-COLOR. These are atomic, i.e. only involve variables and constants taken in $\mathcal{L}$.

7.4 Related Work

In previous work with Sylvain Conchon [20], we proposed a generic approach to information flow analysis, based on a suitable colored semantics, where terms are annotated with security colors taken from $\mathcal{L}$. (See e.g. Sewell and Vitek’s colored box-$\pi$-calculus [22].) We suggested, at the time, that this approach should be applicable to the $\pi$-calculus. Later experiments confirmed our intuition, but showed that it naturally leads to a may-testing-based noninterference result. Our attempts to adapt it to a bisimulation setting lead us to the present formulation, where brackets replace colors, allowing a simple bisimulation proof.

Hennessy and Riely’s system [9] shares several basic mechanisms, such as the use of security annotations on processes, channels and judgements, with ours. Instead of using channel types annotated with polarities à la Pierce and Sangiorgi, they use sets of so-called read or write “capabilities”; this seems only a superficial difference. More importantly, they study the asynchronous $\pi$-calculus under may-testing equivalence, which means that information only flows from senders to receivers and allows a channel to be read at a higher security level than it was written. (Contrast this with our discussion of Sect. 7.1.) Lastly, their boxing construct $\rho[P]$ is not analogous to our coloring construct $l : P$. (Compare the corresponding typing rules.) It is used with different meanings in their “resource control” system and in their “information flow” system, which creates a tension and seems to make their noninterference statement a bit awkward.

Sewell and Vitek [22] develop a type system similar to ours. (Their annotations are sets of principals, whereas we employ a slightly more abstract notion of security level.) They do not prove a noninterference result; instead, they state a so-called “causal flow” property. We view this as a serious shortcoming: it is difficult to determine exactly which notion of causality the property reflects. The same criticism can be held against Bodei et al.’s “no read up/no write down” property [4].

The restriction of Honda et al.’s system [11] to nonlinear types seems essentially identical to our system: judgements are annotated with a security level similar to ours (compare $(\text{Deg}_a)$ with T-SUB), and nonlinear channel types are invariant in their security level. The noninterference results stated in [11, 12, 27] also rely on a form of weak bisimulation. They propose the most advanced systems to date; whether our technique can be modified to establish their soundness is an interesting issue.

The problem of noninterference in multi-threaded imperative languages is similar to the one studied here, and has been investigated by several researchers [23, 24, 5]. We believe that our proof technique could be re-used in their setting, allowing the candidate bisimulation relation to be defined implicitly in terms of a type system for a “bracket” calculus – as done here – rather than explicitly, which is clumsier (see e.g. [5]).

Acknowledgements

The paper owes much to numerous discussions with Sylvain Conchon. Alan Schmitt’s diagram-chasing skills were also most helpful.

References


