A Simple View of Type-Secure Information Flow in the π-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 π-calculus. This is done by reducing the problem to* subject reduction *– a safety property – for a nonstandard, but fairly natural, extension of the π-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. [19]. Recently, researchers [9, 11, 21, 10, 12, 26] have begun further extending this study to low-level, concurrent calculi, such as the π-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 [21] 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, 26] 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 [25], due to the need to keep track of liveness properties. Furthermore, the type systems presented in [12, 26] 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 accept every π-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 Vitek's [21], 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 π-calculus. It is inspired by previous joint work with Vincent Simonet [20].

Our calculus, baptized the $\langle\pi\rangle$-calculus, describes the independent execution of a *pair* of processes (which we arbitrarily index by $\{1,2\}$), while keeping track of their *shared* sub-processes. For instance, the $\langle\pi\rangle$-calculus process $\mathbf{P} \mid \langle\mathbf{Q}\rangle_1 \mid \langle\mathbf{R}\rangle_2$ represents the pair $(\mathbf{P} \mid \mathbf{Q}, \mathbf{P} \mid \mathbf{R})$, while explicitly recording the fact that the sub-process $\mathbf{P}$ is shared. A sub-term of the form $\langle\mathbf{P}\rangle_i$ indicates that $\mathbf{P}$ 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 $\bar{x} \mid \langle x.\mathbf{P}\rangle$ would represent the pair of processes $(\bar{x},\ \bar{x} \mid x.\mathbf{P})$, while encoding the fact that $x.\mathbf{P}$ 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 $(\bar{x},\ \mathbf{P})$, 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 π-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 $\mathbf{P}$ to $\langle\mathbf{P}\rangle_1 \mid \langle\mathbf{P}\rangle_2$. 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 π-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 π-calculus [18], but is extended with *security annotations*, a standard notion in information flow analyses; see e.g. [11, 21]. 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 π-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 $\mathcal{N}$ of* names*. Let $\tilde{y}, \tilde{z}, \ldots$ 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*

$$N ::= x(\tilde{y}).P \mid \bar{x}\langle\tilde{z}\rangle.P \mid \mathbf{0} \mid N + N$$
$$P ::= N \mid (P \mid P) \mid !P \mid \nu x.P \mid \langle P\rangle_i$$

$x(\tilde{y}).P$ binds $\tilde{y}$ in P, and $\nu x.P$ binds x in P. From this information, the usual notions of free names, capture-free substitution, and α-convertibility are deduced. We write $\mathrm{fn}(P)$ for the set of free names of a process P. We identify processes up to α-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 $\langle\mathbf{P}\rangle_i$, i.e. we never nest brackets.*

A single $\langle\pi\rangle$-calculus process is meant to represent a *pair* of standard π-calculus processes. In particular, $\langle\mathbf{P}\rangle_1$ stands for the pair $(\mathbf{P}, \mathbf{0})$, while $\langle\mathbf{P}\rangle_2$ stands for $(\mathbf{0}, \mathbf{P})$. A $\langle\pi\rangle$-calculus process of the form $\mathbf{P}$ represents *shared* structure: it stands for the pair $(\mathbf{P}, \mathbf{P})$. More generally, an arbitrary process P stands for the pair $(\pi_1(P), \pi_2(P))$, where the projection functions π_1 and π_2 are defined as follows.

Definition 2 *Let $\{i, j\} = \{1, 2\}$. The i^{th}* projection *function, written π_i, satisfies the laws $\pi_i(\langle\mathbf{P}\rangle_i) = \mathbf{P}$ and $\pi_i(\langle\mathbf{P}\rangle_j) = \mathbf{0}$ and is a homomorphism on other (i.e. standard) process forms. We often write $\pi_i P$ for $\pi_i(P)$. For all P, $\pi_i P$ 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 $P \leq_0 P \mid \mathbf{0}$.*

(By *compatible*, we mean closed under all contexts.) Viewing π_i as a relation, we will write $P \;\pi_i \cdot \geq_0^\star\; \mathbf{P}$ or $\mathbf{P} \;\leq_0^\star \cdot \pi_i^{-1}\; P$ to denote $\pi_i P \geq_0^\star \mathbf{P}$. (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 π-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 π-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, compatible relation over processes such that the following laws hold:*

1. *$N + \mathbf{0} \equiv N$, $N \equiv N + \mathbf{0}$, $N_1 + N_2 \equiv N_2 + N_1$, $(N_1 + N_2) + N_3 \equiv N_1 + (N_2 + N_3)$;*
2. *$P \mid \mathbf{0} \equiv P$, $P \equiv P \mid \mathbf{0}$, $P_1 \mid P_2 \equiv P_2 \mid P_1$, $(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$;*
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 $\langle[]\rangle_i$.*

All structural congruence laws are standard. All evaluation contexts are standard as well, except $\langle[]\rangle_i$, which allows reduction under brackets.

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

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:

SPLIT'
$$M \mapsto \langle\pi_1 M\rangle_1 \mid \langle\pi_2 M\rangle_2$$

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,

COMM
$$(M + x(\tilde{y}).P) \mid (N + \bar{x}\langle\tilde{z}\rangle.Q) \mapsto P[\tilde{z}/\tilde{y}] \mid Q$$

EXTR
$$\frac{x \notin \mathrm{fn}(Q)}{(\nu x.P) \mid Q \mapsto \nu x.(P \mid Q)}$$

REPL
$$!P \mapsto P \mid !P$$

NEW
$$\mathbf{0} \mapsto \nu x.\mathbf{0}$$

CONTEXT
$$\frac{P \mapsto P'}{E[P] \mapsto E[P']}$$

SPLIT
$$\frac{\pi_i M \# \mathbf{N} \qquad \{i, j\} = \{1, 2\}}{M \mid \langle\mathbf{N}\rangle_i \mapsto \langle\pi_i M \mid \mathbf{N}\rangle_i \mid \langle\pi_j M\rangle_j}$$

GLUE
$$\frac{\mathbf{M} \# \mathbf{N}}{\langle\mathbf{M}\rangle_i \mid \langle\mathbf{N}\rangle_i \mapsto \langle\mathbf{M} \mid \mathbf{N}\rangle_i}$$

BREAK
$$\langle\mathbf{P} \mid \mathbf{Q}\rangle_i \mapsto \langle\mathbf{P}\rangle_i \mid \langle\mathbf{Q}\rangle_i$$

PUSH
$$\langle\nu x.\mathbf{P}\rangle_i \mapsto \nu x.\langle\mathbf{P}\rangle_i$$

Figure 1. Semantics of the $\langle\pi\rangle$-calculus

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, 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 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 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 $\mathbf{N}$ which already appears under a bracket $\langle\mathbf{N}\rangle_i$.

Rules GLUE and BREAK can be understood as restrictions of a (hypothetical) structural congruence rule:

$$\langle\mathbf{P} \mid \mathbf{Q}\rangle_i \equiv \langle\mathbf{P}\rangle_i \mid \langle\mathbf{Q}\rangle_i$$

Again, allowing this equivalence to hold would be correct as far as the untyped semantics is concerned, but would pose a slight technical typing problem. The premise in rule 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 GLUE for more details.)

Lastly, rule PUSH allows pushing brackets down inside ν binders. In conjunction with BREAK, this lets them move down to the level of normal processes, where they can be dealt with by SPLIT.

BREAK and PUSH may look surprising at first sight, because they appear to artificially create shared structure: a parallel composition operator $\mid$ (resp. a binder νx) is moved *outside* of a bracket. To see why this is sound, consider the j^{th} projections of these rules, where $\{i, j\} = \{1, 2\}$. They are $\mathbf{0} \Rightarrow \mathbf{0} \mid \mathbf{0}$ and $\mathbf{0} \Rightarrow \nu x.\mathbf{0}$, i.e. still correct reduction steps. This remark is formalized by Lemma 2 below.

3.3 Relating the $\langle\pi\rangle$-Calculus to the π-Calculus

We now relate the $\langle\pi\rangle$-calculus to the π-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 π-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 congruence (resp. reduction) step is a correct congruence (resp. reduction) step as well.

Lemma 1 *Let $i \in \{1, 2\}$. If $P \equiv P'$, then $\pi_i P \equiv \pi_i P'$.*

Proof. By induction on the derivation of $P \equiv P'$. If it is an instance of one of the laws in Definition 4, then the result is a direct consequence of the fact that π_i behaves homomorphically on standard process forms. Furthermore, the congruence axioms are also preserved by projection: this holds (again) by homomorphism in all standard cases, and by projection to either $\mathbf{P} \equiv \mathbf{Q}$ or $\mathbf{0} \equiv \mathbf{0}$ in the case where $\langle\mathbf{P}\rangle_j \equiv \langle\mathbf{Q}\rangle_j$ stems from $\mathbf{P} \equiv \mathbf{Q}$. □

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

Proof. By induction on the derivation of $P \mapsto P'$.

Case COMM. π_i is a homomorphism on all process forms involved. $(\pi_i P)[\tilde{z}/\tilde{y}]$ is $\pi_i(P[\tilde{z}/\tilde{y}])$. The result follows by COMM.

Case EXTR. π_i is a homomorphism on all process forms involved. The result follows by EXTR, noticing that $x \notin \mathrm{fn}(Q)$ implies $x \notin \mathrm{fn}(\pi_i Q)$.

Case REPL. Again, π_i is a homomorphism on all process forms involved.

Case NEW. Immediate.

Case CONTEXT. The three sub-cases are $E = [] \mid Q$, $E = \nu x.[]$ and $E = \langle[]\rangle_j$. The first two are easily dealt with by appealing to the induction hypothesis, applying CONTEXT and using π_i's homomorphic behavior. Let us focus on the third sub-case, where P and P' are respectively of the form $\langle \mathbf{Q} \rangle_j$ and $\langle \mathbf{Q}' \rangle_j$, and $\mathbf{Q} \mapsto \mathbf{Q}'$ holds. If $i \neq j$, then $\pi_i P = \pi_i P' = \mathbf{0}$ and the result is immediate. If $i = j$, then $\pi_i P = \mathbf{Q}$ and $\pi_i P' = \mathbf{Q}'$, so $\pi_i P$ reduces to $\pi_i P'$ as desired.

Case SPLIT. The image of the reduction through π_i is either $\pi_i M \mid \mathbf{N} \Rightarrow (\pi_i M \mid \mathbf{N}) \mid \mathbf{0}$ or $\pi_i M \mid \mathbf{0} \Rightarrow \mathbf{0} \mid \pi_i M$. Both hold.

Case GLUE. The image of the reduction through π_i is either $\mathbf{M} \mid \mathbf{N} \Rightarrow \mathbf{M} \mid \mathbf{N}$ or $\mathbf{0} \mid \mathbf{0} \Rightarrow \mathbf{0}$. Both hold.

Case BREAK. The image of the reduction through π_i is either $\mathbf{P} \mid \mathbf{Q} \Rightarrow \mathbf{P} \mid \mathbf{Q}$ or $\mathbf{0} \Rightarrow \mathbf{0} \mid \mathbf{0}$. Both hold.

Case PUSH. The image of the reduction through π_i is either $\nu x.\mathbf{P} \Rightarrow \nu x.\mathbf{P}$ or $\mathbf{0} \Rightarrow \nu x.\mathbf{0}$. Both hold. □

Lemma 3 *Let $i \in \{1, 2\}$. If $P \Rightarrow P'$, then $\pi_i P \Rightarrow \pi_i P'$.*

Proof. Direct consequence of Lemmas 1 and 2. □

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 $\mathbf{P} \mid \mathbf{0}$, which reduces to $\mathbf{P}$ via a structural congruence step, then P may be of the form $\mathbf{P} \mid \langle \mathbf{Q} \rangle_j$, 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 $\leq_0^\star$ introduced in Definition 3.

We will also require another auxiliary pre-order, which "complements" $\leq_0$ with respect to structural congruence. It is defined as follows. (Note that $\leq_0 \cup \leqq_0$ is $\equiv$.)

Definition 7 *Let $\leqq_0$ be the smallest reflexive, compatible relation over processes which enjoys all laws of Definition 4 except $P \leqq_0 P \mid \mathbf{0}$.*

Our first lemma states that every standard structural congruence step, except the removal of extra $\mathbf{0}$ processes, can be emulated by the $\langle \pi \rangle$-calculus. The careful reader will notice that this lemma is the reason why we turn scope extrusion and replication into reduction rules. Indeed, when used in the reverse direction, neither of these rules can, in general, be emulated. Scope intrusion fails because a $\langle \pi \rangle$-calculus process has more free names than its projection. Replication folding fails because two distinct $\langle \pi \rangle$-calculus processes may have identical projections.

Lemma 4 *Let $i \in \{1, 2\}$. If $\mathbf{P} \geqq_0 \mathbf{P}'$ and $\mathbf{P} = \pi_i P$, then there exists some process P' such that $P \Rightarrow P'$ and $\mathbf{P}' = \pi_i P'$.*

Proof. By induction on the derivation of $\mathbf{P} \geqq_0 \mathbf{P}'$. If $P = \langle \mathbf{P} \rangle_i$, then P is structurally congruent to $\langle \mathbf{P}' \rangle_i$, whose i^{th} projection is $\mathbf{P}'$. Thus, we will silently omit this sub-case in all cases below. Let $\{i, j\} = \{1, 2\}$.

Case $\mathbf{P} = \mathbf{M} + \mathbf{0} \geqq_0 \mathbf{M} = \mathbf{P}'$. Then, P is $M + N$, where $\pi_i M = \mathbf{M}$ and $\pi_i N = \mathbf{0}$. Furthermore, N must be $\mathbf{0}$, because no other normal term has $\mathbf{0}$ as its projection. Thus, $P = M + \mathbf{0}$ is structurally congruent to M, whose i^{th} projection is $\mathbf{M} = \mathbf{P}'$.

Case $\mathbf{P} = \mathbf{M} \geqq_0 \mathbf{M} + \mathbf{0} = \mathbf{P}'$. Take $P' = P + \mathbf{0}$.

Case $\mathbf{P} = \mathbf{M}_1 + \mathbf{M}_2 \geqq_0 \mathbf{M}_2 + \mathbf{M}_1 = \mathbf{P}'$. Then, P is $M_1 + M_2$, where $\pi_i M_k = \mathbf{M}_k$ for $k \in \{1, 2\}$. Take $P' = M_2 + M_1$.

Case $\mathbf{P} = \mathbf{M}_1 + (\mathbf{M}_2 + \mathbf{M}_3) \geqq_0 (\mathbf{M}_1 + \mathbf{M}_2) + \mathbf{M}_3 = \mathbf{P}'$. Then, P is $M_1 + (M_2 + M_3)$, where $\pi_i M_k = \mathbf{M}_k$ for $k \in \{1, 2, 3\}$. (Indeed, because a summand must be a normal term, no other sub-cases arise.) Take $P' = (M_1 + M_2) + M_3$.

Case $\mathbf{P} = \mathbf{Q} \geqq_0 \mathbf{Q} \mid \mathbf{0} = \mathbf{P}'$. Take $P' = P \mid \mathbf{0}$.

Case $\mathbf{P} = \mathbf{Q}_1 \mid \mathbf{Q}_2 \geqq_0 \mathbf{Q}_2 + \mathbf{Q}_1 = \mathbf{P}'$. Then, P is $Q_1 \mid Q_2$, where $\pi_i Q_k = \mathbf{Q}_k$ for $k \in \{1, 2\}$. Take $P' = Q_2 \mid Q_1$.

Case $\mathbf{P} = \mathbf{Q}_1 \mid (\mathbf{Q}_2 \mid \mathbf{Q}_3) \geqq_0 (\mathbf{Q}_1 \mid \mathbf{Q}_2) \mid \mathbf{Q}_3 = \mathbf{P}'$. Then, either P is $Q_1 \mid (Q_2 \mid Q_3)$, where $\pi_i Q_k = \mathbf{Q}_k$ for $k \in \{1, 2, 3\}$, or P is $Q_1 \mid \langle \mathbf{Q}_2 \mid \mathbf{Q}_3 \rangle_i$, where $\pi_i Q_1 = \mathbf{Q}_1$. If the former, take $P' = (Q_1 \mid Q_2) \mid Q_3$. If the latter, then BREAK reduces P to $Q_1 \mid (\langle \mathbf{Q}_2 \rangle_i \mid \langle \mathbf{Q}_3 \rangle_i)$, and we are back to the previous sub-case.

Case $\mathbf{P} = \nu x.\nu y.\mathbf{Q} \geqq_0 \nu y.\nu x.\mathbf{Q} = \mathbf{P}'$. Then, P is either $\nu x.\nu y.Q$, where $\pi_i Q = \mathbf{Q}$, or $\nu x.\langle \nu y.\mathbf{Q} \rangle_i$. If the former, take $P' = \nu y.\nu x.Q$. If the latter, then, by PUSH, P reduces to $\nu x.\nu y.\langle \mathbf{Q} \rangle_i$, which brings us back to the previous sub-case.

Case $\mathbf{P} = C[\mathbf{P}_0] \geqq_0 C[\mathbf{P}_0'] = \mathbf{P}'$, where $\mathbf{P}_0 \geqq_0 \mathbf{P}_0'$ and C is some (standard) process constructor. Then, P must of the form $C[P_0]$, where $\pi_i P_0 = \mathbf{P}_0$. By induction hypothesis, there exists some P_0' such that $P_0 \Rightarrow P_0'$ and $\mathbf{P}_0' = \pi_i P_0'$. These entail $P = C[P_0] \Rightarrow C[P_0']$ and $\mathbf{P}' = C[\mathbf{P}_0'] = C[\pi_i P_0'] = \pi_i C[P_0']$. □

The second main lemma states that every standard reduction step can be emulated (again, possibly introducing some extra $\mathbf{0}$'s in parallel) by the $\langle \pi \rangle$-calculus.

Lemma 5 *Let $i \in \{1, 2\}$. If $\mathbf{P} \mapsto \mathbf{P}'$ and $\mathbf{P} = \pi_i P$, then there exists some process P' such that $P \Rightarrow P'$ and $\mathbf{P}' \leq_0 \pi_i P'$.*

Proof. By induction on the derivation of $\mathbf{P} \mapsto \mathbf{P}'$. Let $\{i, j\} = \{1, 2\}$.

Case COMM. The reduction is

$$\begin{array}{rcl} \mathbf{P} & = & (\mathbf{N}_1 + x(\tilde{y}).\mathbf{P}_1) \mid (\mathbf{N}_2 + \bar{x}\langle \tilde{z} \rangle.\mathbf{P}_2) \\ & \mapsto & \mathbf{P}_1[\tilde{z}/\tilde{y}] \mid \mathbf{P}_2 = \mathbf{P}' \end{array}$$

Which processes P have $\mathbf{P}$ as their i^{th} projection? Projection preserves structure, except it may discard some $\langle \cdot \rangle_i$ constructors. So, in P, such a constructor may enclose the outermost $\mid$ node. If not, then each of the two $+$ nodes may (or may not) be enclosed by such a constructor. (The summands $x(\tilde{y}).\mathbf{P}_1$ and $\bar{x}\langle\tilde{z}\rangle.\mathbf{P}_2$ may not be enclosed in such a way, because a summand must be a normal process.) This observation gives rise to five sub-cases.

1. If the outermost node is enclosed, then P is $\langle\mathbf{P}\rangle_i$. Because $\mathbf{P} \mapsto \mathbf{P}'$ holds, so does $\langle\mathbf{P}\rangle_i \mapsto \langle\mathbf{P}'\rangle_i$, by CONTEXT. Furthermore, $\langle\mathbf{P}'\rangle_i$ has $\mathbf{P}'$ as its i^{th} projection.

2. If both sum nodes are enclosed, then, because $\mathbf{P}$ is reducible, GLUE applies, reducing P to $\langle\mathbf{P}\rangle_i$. We fall back to the previous sub-case.

3. If the right-hand sum alone is enclosed, then P is $(N_1 + x(\tilde{y}).P_1) \mid \langle\mathbf{N}_2 + \bar{x}\langle\tilde{z}\rangle.\mathbf{P}_2\rangle_i$, where $\pi_i N_1 = \mathbf{N}_1$ and $\pi_i P_1 = \mathbf{P}_1$. Because $\mathbf{P}$ is reducible, SPLIT applies, showing that P reduces to

$$\begin{array}{ll} & \langle(\mathbf{N}_1 + x(\tilde{y}).\mathbf{P}_1) \mid (\mathbf{N}_2 + \bar{x}\langle\tilde{z}\rangle.\mathbf{P}_2)\rangle_i \\ \mid & \langle(\pi_j N_1 + x(\tilde{y}).\pi_j P_1)\rangle_j \end{array}$$

   By COMM and CONTEXT, this in turn reduces to

$$\langle\mathbf{P}_1[\tilde{z}/\tilde{y}] \mid \mathbf{P}_2\rangle_i \mid \langle(\pi_j N_1 + x(\tilde{y}).\pi_j P_1)\rangle_j$$

   whose i^{th} projection is $(\mathbf{P}_1[\tilde{z}/\tilde{y}] \mid \mathbf{P}_2) \mid \mathbf{0} = \mathbf{P}' \mid \mathbf{0} \geq_0 \mathbf{P}'$.

4. The sub-case where the left-hand sum alone is enclosed is symmetric.

5. If no sum node is enclosed, then P is

$$(N_1 + x(\tilde{y}).P_1) \mid (N_2 + \bar{x}\langle\tilde{z}\rangle.P_2)$$

   where $\pi_i N_k = \mathbf{N}_k$ and $\pi_i P_k = \mathbf{P}_k$ for $k \in \{1,2\}$. Then, by COMM, P reduces to $P_1[\tilde{z}/\tilde{y}] \mid P_2$, whose i^{th} projection is precisely $\mathbf{P}'$.

Case EXTR. The reduction is

$$\mathbf{P} = (\nu x.\mathbf{Q}_1) \mid \mathbf{Q}_2 \mapsto \nu x.(\mathbf{Q}_1 \mid \mathbf{Q}_2) = \mathbf{P}'$$

where $x \notin \text{fn}(\mathbf{Q}_2)$. By BREAK and PUSH, P must reduce to some process of the form $(\nu x.Q_1) \mid Q_2$, where $\pi_i Q_k = \mathbf{Q}_k$ for $k \in \{1,2\}$. Pick some name $z \notin \text{fn}(\nu x.Q_1) \cup \text{fn}(Q_2)$. Then, P may be written $(\nu z.Q_1[z/x]) \mid Q_2$, which, by EXTR, reduces to $P' = \nu z.(Q_1[z/x] \mid Q_2)$. We have $\pi_i P' = \nu z.(\mathbf{Q}_1[z/x] \mid \mathbf{Q}_2) = \mathbf{P}'$.

Case REPL. The reduction is

$$\mathbf{P} = !\mathbf{Q} \mapsto \mathbf{Q} \mid !\mathbf{Q} = \mathbf{P}'$$

Two sub-cases arise: either P is $\langle\mathbf{P}\rangle_i$, or P is $!Q$, where $\pi_i Q = \mathbf{Q}$. If the former, then P reduces to $\langle\mathbf{P}'\rangle_i$. If the latter, then P reduces to $Q \mid !Q$. In either case, the reduct has $\mathbf{P}'$ as its i^{th} projection.

Case NEW. The reduction is $\mathbf{P} = \mathbf{0} \mapsto \nu x.\mathbf{0} = \mathbf{P}'$. Because terms are identified modulo α-conversion, we may assume $x \notin \text{fn}(P)$. Take $P' = \nu x.P$. It is easy to check that $P \Rightarrow P'$ holds. Furthermore, we have $\pi_i P' = \nu x.\mathbf{0} = \mathbf{P}'$.

Case CONTEXT. The reduction is

$$\mathbf{P} = \mathbf{E}[\mathbf{Q}] \mapsto \mathbf{E}[\mathbf{Q}'] = \mathbf{P}'$$

where $\mathbf{Q} \mapsto \mathbf{Q}'$. Because $\mathbf{E}$ must be standard, only two sub-cases arise.

1. Sub-case $\mathbf{E} = [] \mid \mathbf{Q}_0$. Then, P is either $\langle\mathbf{P}\rangle_i$, or $Q \mid Q_0$, where $\pi_i Q = \mathbf{Q}$ and $\pi_i Q_0 = \mathbf{Q}_0$. If the former, take $P' = \langle\mathbf{P}'\rangle_i$. If the latter, then the induction hypothesis yields Q' such that $Q \Rightarrow Q'$ and $\mathbf{Q}' \leq_0 \pi_i Q'$. Let $P' = Q' \mid Q_0$. Then, by CONTEXT, $P \Rightarrow P'$ holds. Furthermore, $\mathbf{P}' = \mathbf{E}[\mathbf{Q}'] \leq_0 \mathbf{E}[\pi_i Q'] = \pi_i P'$.

2. Sub-case $\mathbf{E} = \nu x.[]$ is dealt with in a similar way. □

Lemma 6 *If $\mathbf{P} \mapsto \mathbf{P}'$ and $\mathbf{P} \leq_0 \mathbf{Q}$, then there exists some process $\mathbf{Q}'$ such that $\mathbf{Q} \geqq_0^\star \cdot \mapsto \mathbf{Q}'$ and $\mathbf{P}' \leq_0^\star \mathbf{Q}'$.*

Proof. (Sketch.) By induction on the derivation of $\mathbf{P} \mapsto \mathbf{P}'$.

Case COMM. If the $\mathbf{0}$ process is added inside one of the summands, then it does not prevent reduction. If it is added somewhere above the $+$ operators, then it can be brought to the top level using commutativity and associativity of parallel composition; then, reduction can take place under this context.

Case EXTR, REPL, NEW. Similar. (The only case where we actually end up with $\mathbf{P}' \leq_0^+ \mathbf{Q}'$, rather than $\mathbf{P}' \leq_0 \mathbf{Q}'$, is that of REPL, where the $\mathbf{0}$ process may be added under a replication operator.)

Case CONTEXT. If the $\mathbf{0}$ process is added inside E, then it does not prevent reduction. If it is added inside P, then the induction hypothesis can be applied. □

For technical reasons, we need to specialize lemmas 5 and 6 in the particular case where the reduction at hand is an instance of scope extrusion. This is done in the following definition and lemmas.

Definition 8 *Let $\overset{\nu}{\mapsto}$ be the relation generated by* EXTR *and* CONTEXT.

Lemma 7 *Let $i \in \{1,2\}$. If $\mathbf{P} \overset{\nu}{\mapsto} \mathbf{P}'$ and $\mathbf{P} = \pi_i P$, then there exists some process P' such that $P \Rightarrow P'$ and $\mathbf{P}' = \pi_i P'$.*

Proof. Extract the two relevant cases out of the proof of Lemma 5. □

Lemma 8 *If $\mathbf{P} \overset{\nu}{\mapsto} \mathbf{P}'$ and $\mathbf{P} \leq_0 \mathbf{Q}$, then there exists some process $\mathbf{Q}'$ such that $\mathbf{Q} \geqq_0^\star \cdot \overset{\nu}{\mapsto} \mathbf{Q}'$ and $\mathbf{P}' \leq_0 \mathbf{Q}'$.*

Proof. Extract the two relevant cases out of the proof of Lemma 6. □

The careful reader will notice that the following lemma is the reason why we turn creation of new bound names into a reduction rule. Indeed, when used in the reverse direction, the rule cannot be emulated: if $\nu x.\mathbf{0}$ moves to $\mathbf{0}$, $\nu x.(\mathbf{0} \mid \mathbf{0})$ cannot follow step. (Recall that scope intrusion is not allowed.)

Lemma 9 *If* $\mathbf{P} \geqq_0 \mathbf{P}'$ *and* $\mathbf{P} \leq_0 \mathbf{Q}$*, then there exists some process* $\mathbf{Q}'$ *such that* $\mathbf{Q}\ (\geqq_0 \cup \overset{\nu}{\mapsto})^\star\ \mathbf{Q}'$ *and* $\mathbf{P}' \leq_0 \mathbf{Q}'$.

Proof. (Sketch.) By induction on the derivation of $\mathbf{P} \geqq_0 \mathbf{P}'$.

Case $+$-monoid law. If the $\mathbf{0}$ process is added inside one of the summands, then it does not block reduction. Otherwise, it must added at top level; reduction can then take place under this context.

Case $|$-monoid law. If the $\mathbf{0}$ process is added inside one of the components, then it does not block reduction. Otherwise, it can be brought to the top level using commutativity and associativity of parallel composition; reduction can then take place under this context.

Case $\nu x.\nu y.\mathbf{Q} \geqq_0 \nu y.\nu x.\mathbf{Q}$. If the $\mathbf{0}$ process is added inside both ν binders, then it does not block reduction. Otherwise, it can be brought down to this position using scope extrusion.

Case congruence law. If the $\mathbf{0}$ process is added inside C, then it does not prevent reduction. If it is added inside P, then the induction hypothesis can be applied. □

The last two lemmas yield the following:

Lemma 10 *If* $\mathbf{P}\ (\geqq_0 \cup \overset{\nu}{\mapsto})^\star\ \mathbf{P}'$ *and* $\mathbf{P} \leq_0 \mathbf{Q}$*, then there exists some process* $\mathbf{Q}'$ *such that* $\mathbf{Q}\ (\geqq_0 \cup \overset{\nu}{\mapsto})^\star\ \mathbf{Q}'$ *and* $\mathbf{P}' \leq_0 \mathbf{Q}'$.

Proof. Direct consequence of Lemmas 8 and 9. □

This in turns yields:

Lemma 11 *If* $\mathbf{P}\ (\geqq_0 \cup \overset{\nu}{\mapsto})^\star\ \mathbf{P}'$ *and* $\mathbf{P} \leq_0^n \mathbf{Q}$*, then there exists some process* $\mathbf{Q}'$ *such that* $\mathbf{Q}\ (\geqq_0 \cup \overset{\nu}{\mapsto})^\star\ \mathbf{Q}'$ *and* $\mathbf{P}' \leq_0^n \mathbf{Q}'$.

Proof. Direct consequence of Lemma 10. □

We may now proceed as follows.

Lemma 12 *If* $\mathbf{P} \mapsto \mathbf{P}'$ *and* $\mathbf{P} \leq_0^\star \mathbf{Q}$*, then there exists some process* $\mathbf{Q}'$ *such that* $\mathbf{Q}\ (\geqq_0 \cup \overset{\nu}{\mapsto})^\star \cdot \mapsto \mathbf{Q}'$ *and* $\mathbf{P}' \leq_0^\star \mathbf{Q}'$.

Proof. Direct consequence of Lemmas 6 and 11. □

Lemma 13 *Let* $i \in \{1,2\}$*. If* $\mathbf{P} \mapsto \mathbf{P}'$ *and* $\mathbf{P} \leq_0^\star \cdot \pi_i^{-1}\ P$*, then there exists some process* P' *such that* $P \Rightarrow P'$ *and* $\mathbf{P}' \leq_0^\star \cdot \pi_i^{-1}\ P'$.

Proof. Direct consequence of Lemmas 12, 4, 7 and 5. □

Lemma 14 *Let* $i \in \{1,2\}$*. If* $\mathbf{P} \geq_0 \mathbf{P}'$ *and* $\mathbf{P} \leq_0^\star \cdot \pi_i^{-1}\ P$*, then there exists some process* P' *such that* $P \Rightarrow P'$ *and* $\mathbf{P}' \leq_0^\star \cdot \pi_i^{-1}\ P'$.

Proof. Immediate (take $P' = P$). □

Lemma 15 *Let* $i \in \{1,2\}$*. If* $\mathbf{P} \geqq_0 \mathbf{P}'$ *and* $\mathbf{P} \leq_0^\star \cdot \pi_i^{-1}\ P$*, then there exists some process* P' *such that* $P \Rightarrow P'$ *and* $\mathbf{P}' \leq_0^\star \cdot \pi_i^{-1}\ P'$.

Proof. Direct consequence of Lemmas 11, 4 and 7. □

Lemma 16 *Let* $i \in \{1,2\}$*. If* $\mathbf{P} \Rightarrow \mathbf{P}'$ *and* $\mathbf{P} \leq_0^\star \cdot \pi_i^{-1}\ P$*, then there exists some process* P' *such that* $P \Rightarrow P'$ *and* $\mathbf{P}' \leq_0^\star \cdot \pi_i^{-1}\ P'$.

Proof. Direct consequence of Lemmas 13, 14 and 15. □

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 type system for the π-calculus – namely Pierce and Sangiorgi's [18], which we choose for its simplicity – with security annotations. Its typing judgements are of the form $\Gamma \vdash_{\mathrm{pc}} P$, 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* Γ*,* P *is well-typed and will affect only observers of security clearance* pc *or higher*. It may also be read: *under assumptions* Γ*, 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 by low-level observers. To achieve this, our type system will guarantee that processes of the form $\langle\mathbf{P}\rangle_i$ – which we use to encode the differences between two processes – can affect only high-security-level observers. In other words, for $\Gamma \vdash_{\mathrm{pc}} \langle\mathbf{P}\rangle_i$ to hold, we will require $\mathrm{pc} \notin L$. (See rule T-BRACKET in Fig. 2.) This will be our only use of L in this section.

As in [18], every channel type t carries a polarity p which tells whether the channel may be used for input, output, or both. It is further annotated with a security level $l \in \mathcal{L}$, which tells how much information may be obtained by successfully reading from or writing to the channel.

Definition 9 *A* polarity p *is one of* $\{-, +, \pm\}$. Types t *are of the form* $\langle \tilde{t} \rangle_l^p$.

Definition 10 *Polarities are ordered by* $\pm \le -$, $\pm \le +$. *Types are ordered by*

$$\frac{p \le p' \qquad (p' \le - \Rightarrow \tilde{t} \le \tilde{t'}) \qquad (p' \le + \Rightarrow \tilde{t'} \le \tilde{t})}{\langle \tilde{t} \rangle_l^p \le \langle \tilde{t'} \rangle_l^{p'}}$$

(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 20, which itself plays a key role in establishing subject reduction.

Definition 11 *The type system of the* $\langle\pi\rangle$*-calculus is defined by Fig. 2. It involves two separate judgement forms. Judgements of the form* $\Gamma \vdash_{(\mathrm{pc})} N$ *concern normal processes. Judgements of the form* $\Gamma \vdash_{\mathrm{pc}} P$ *concern arbitrary processes.*

$$\textsc{T-Recv}\quad \frac{\Gamma(x) \le \langle \tilde{t} \rangle^-_{\mathrm{pc}} \qquad \Gamma; \tilde{y} : \tilde{t} \vdash_{\mathrm{pc}} P}{\Gamma \vdash_{(\mathrm{pc})} x(\tilde{y}).P}$$

$$\textsc{T-Send}\quad \frac{\Gamma(x) \le \langle \tilde{t} \rangle^+_{\mathrm{pc}} \qquad \Gamma(\tilde{y}) \le \tilde{t} \qquad \Gamma \vdash_{\mathrm{pc}} P}{\Gamma \vdash_{(\mathrm{pc})} \bar{x}\langle \tilde{y} \rangle.P} \qquad \textsc{T-Null}\quad \Gamma \vdash_{(\mathrm{pc})} \mathbf{0}$$

$$\textsc{T-Sum}\quad \frac{\Gamma \vdash_{(\mathrm{pc})} M \qquad \Gamma \vdash_{(\mathrm{pc})} N}{\Gamma \vdash_{(\mathrm{pc})} M + N} \qquad \textsc{T-Normal}\quad \frac{\Gamma \vdash_{(\mathrm{pc})} N}{\Gamma \vdash_{\mathrm{pc}} N}$$

$$\textsc{T-Par}\quad \frac{\Gamma \vdash_{\mathrm{pc}} P \qquad \Gamma \vdash_{\mathrm{pc}} Q}{\Gamma \vdash_{\mathrm{pc}} P \mid Q} \qquad \textsc{T-Repl}\quad \frac{\Gamma \vdash_{\mathrm{pc}} P}{\Gamma \vdash_{\mathrm{pc}} !P} \qquad \textsc{T-New}\quad \frac{\Gamma; x : t \vdash_{\mathrm{pc}} P}{\Gamma \vdash_{\mathrm{pc}} \nu x.P}$$

$$\textsc{T-Bracket}\quad \frac{\Gamma \vdash_{\mathrm{pc}} \mathbf{P} \qquad \mathrm{pc} \notin L}{\Gamma \vdash_{\mathrm{pc}} \langle \mathbf{P} \rangle_i} \qquad \textsc{T-Sub}\quad \frac{\Gamma \vdash_{\mathrm{pc}} P \qquad \mathrm{pc}' \le \mathrm{pc}}{\Gamma \vdash_{\mathrm{pc}'} P}$$

Figure 2. Typing rules of the $\langle\pi\rangle$-calculus

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 $\Gamma \vdash_{(\mathrm{pc})} N$ 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 $\Gamma \vdash_{\mathrm{pc}} P$; applying it to judgements of the form $\Gamma \vdash_{(\mathrm{pc})} N$ 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 17 *For* $i \in \{1, 2\}$, $\Gamma \vdash_{\mathrm{pc}} P$ *implies* $\Gamma \vdash_{\mathrm{pc}} \pi_i P$.

Lemma 18 $\Gamma \vdash_{\mathrm{pc}} P$ *and* $P \equiv P'$ *imply* $\Gamma \vdash_{\mathrm{pc}} P'$.

Lemma 19 $\Gamma; \tilde{y} : \tilde{t} \vdash_{\mathrm{pc}} P$ *and* $\Gamma(\tilde{z}) \le \tilde{t}$ *imply* $\Gamma \vdash_{\mathrm{pc}} P[\tilde{z}/\tilde{y}]$.

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 Γ, 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 20 *Assume* $\Gamma \vdash_{(\mathrm{pc}_1)} M$ *and* $\Gamma \vdash_{(\mathrm{pc}_2)} N$*. If* $M \mid N \mapsto R$*, then* $\Gamma \vdash_{\mathrm{pc}} R$ *holds, where* $\mathrm{pc} = \mathrm{pc}_1 = \mathrm{pc}_2$.

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

By T-SUM and T-RECV, the first hypothesis yields $\Gamma \vdash_{(\mathrm{pc}_1)} M'$, $\Gamma(x) \leq \langle \tilde{t_1} \rangle^-_{\mathrm{pc}_1}$ and $\Gamma; \tilde{y} : \tilde{t_1} \vdash_{\mathrm{pc}_1} P$. By T-SUM and T-SEND, the second one yields $\Gamma \vdash_{(\mathrm{pc}_2)} N'$, $\Gamma(x) \leq \langle \tilde{t_2} \rangle^+_{\mathrm{pc}_2}$, $\Gamma(\tilde{z}) \leq \tilde{t_2}$ and $\Gamma \vdash_{\mathrm{pc}_2} Q$.

From $\Gamma(x) \leq \langle \tilde{t_1} \rangle^-_{\mathrm{pc}_1}$ and $\Gamma(x) \leq \langle \tilde{t_2} \rangle^+_{\mathrm{pc}_2}$, we deduce $\mathrm{pc}_1 = \mathrm{pc}_2$ and $\tilde{t_2} \leq \tilde{t_1}$. Take $\mathrm{pc} = \mathrm{pc}_1 = \mathrm{pc}_2$. By transitivity of subtyping, $\Gamma(\tilde{z}) \leq \tilde{t_1}$ holds. Lemma 19 then yields $\Gamma \vdash_{\mathrm{pc}_1} P[\tilde{z}/\tilde{y}]$. Using $\mathrm{pc} = \mathrm{pc}_1 = \mathrm{pc}_2$ and T-PAR, we obtain $\Gamma \vdash_{\mathrm{pc}} P[\tilde{z}/\tilde{y}] \mid Q$, that is, $\Gamma \vdash_{\mathrm{pc}} R$. $\square$

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

Lemma 21 (Subject Reduction) $\Gamma \vdash_{\mathrm{pc}} P$ *and* $P \mapsto P'$ *imply* $\Gamma \vdash_{\mathrm{pc}} P'$.

Proof. By induction on the derivation of $P \mapsto P'$.

Case COMM. P is of the form $M \mid N$. By T-SUB, T-PAR and T-NORMAL, this yields $\Gamma \vdash_{(\mathrm{pc}_1)} M$ and $\Gamma \vdash_{(\mathrm{pc}_2)} N$, where $\mathrm{pc} \leq \mathrm{pc}_1$ and $\mathrm{pc} \leq \mathrm{pc}_2$ hold. By Lemma 20 and by T-SUB, $\Gamma \vdash_{\mathrm{pc}} P'$ holds.

Case SPLIT. The reduction is $M \mid \langle \mathbf{N} \rangle_i \mapsto \langle \pi_i M \mid \mathbf{N} \rangle_i \mid \langle \pi_j M \rangle_j$, where $\{i, j\} = \{1, 2\}$ and $\pi_i M \# \mathbf{N}$. Our hypothesis is $\Gamma \vdash_{\mathrm{pc}} M \mid \langle \mathbf{N} \rangle_i$. By T-SUB, T-PAR, T-NORMAL and T-BRACKET, this yields $\Gamma \vdash_{(\mathrm{pc}_1)} M$ and $\Gamma \vdash_{\mathrm{pc}_2} \mathbf{N}$, where $\mathrm{pc} \leq \mathrm{pc}_1$, $\mathrm{pc} \leq \mathrm{pc}_2$, and $\mathrm{pc}_2 \notin L$ 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, $\Gamma \vdash_{(\mathrm{pc}_2)} \mathbf{N}$ holds. $\pi_i M \# \mathbf{N}$ implies $M \# \mathbf{N}$. As a result, we may apply Lemma 20 (either to M and $\mathbf{N}$ or to $\mathbf{N}$ and M), yielding $\mathrm{pc}_1 = \mathrm{pc}_2$.

It follows that $\Gamma \vdash_{\mathrm{pc}_2} M$ holds. By Lemma 17, both $\pi_i M$ and $\pi_j M$ are well-typed under Γ and pc_2. By hypothesis, so is $\mathbf{N}$. Thus, considering $\mathrm{pc}_2 \notin L$, so is $\langle \pi_i M \mid \mathbf{N} \rangle_i \mid \langle \pi_j M \rangle_j$. Because $\mathrm{pc}_2 \geq \mathrm{pc}$, the result follows by T-SUB.

Case GLUE. The reduction is $\langle \mathbf{M} \rangle_i \mid \langle \mathbf{N} \rangle_i \mapsto \langle \mathbf{M} \mid \mathbf{N} \rangle_i$, where $\mathbf{M} \# \mathbf{N}$. Our hypothesis is $\Gamma \vdash_{\mathrm{pc}} \langle \mathbf{M} \rangle_i \mid \langle \mathbf{N} \rangle_i$. As above, this yields $\Gamma \vdash_{(\mathrm{pc}_1)} \mathbf{M}$ and $\Gamma \vdash_{(\mathrm{pc}_2)} \mathbf{N}$, where $\mathrm{pc} \leq \mathrm{pc}_k$ and $\mathrm{pc}_k \notin L$ hold for $k \in \{1, 2\}$. Again, Lemma 20 yields $\mathrm{pc}_1 = \mathrm{pc}_2$. Thus, $\Gamma \vdash_{\mathrm{pc}_2} \langle \mathbf{M} \mid \mathbf{N} \rangle_i$ holds. The result follows by T-SUB.

Case EXTR, REPL, CONTEXT, BREAK, PUSH. Immediate. $\square$

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

$$(M + x(\tilde{y}).P){\downarrow_x} \qquad\qquad (M + \bar{x}\langle \tilde{y} \rangle.P){\downarrow_{\bar{x}}}$$

$$\frac{P{\downarrow_\alpha} \qquad E \text{ does not bind } |\,\alpha\,|}{E[P]{\downarrow_\alpha}}$$

$P{\Downarrow_\alpha}$ *stands for* $(\exists P' \quad P \Rightarrow P' \wedge P'{\downarrow_\alpha})$.

Definition 13 *Let* B *be an arbitrary set of names. A binary relation* $\mathcal{R}$ *over processes is a* weak B-simulation *if and only if*

- $P \mathrel{\mathcal{R}} Q \wedge P \Rightarrow P'$ *implies* $\exists Q' \quad Q \Rightarrow Q' \wedge P' \mathrel{\mathcal{R}} Q'$*; and*
- $|\,\alpha\,| \in B$*,* $P \mathrel{\mathcal{R}} Q$ *and* $P{\Downarrow_\alpha}$ *imply* $Q{\Downarrow_\alpha}$.

$\mathcal{R}$ *is a* weak B-bisimulation *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*-bisimulation. They are* weakly bisimilar *if they are* $\mathcal{N}$*-bisimilar.*

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

Definition 14 *Given a type environment* Γ*,* $\mathrm{low}(\Gamma)$ *denotes the largest set* $B \subseteq \mathcal{N}$ *such that* $x \in B$ *and* $\Gamma(x) = \langle \tilde{t} \rangle^p_l$ *imply* $l \in L$*.* Γ *is said to be an* L-environment *if and only if* $\mathrm{low}(\Gamma) = \mathcal{N}$.

If $x \in \mathrm{low}(\Gamma)$ holds, then, by rule T-BRACKET, no well-typed process observable at x can appear under brackets. So, in that case, observability at x must be preserved by projection. This is expressed by the following simple lemma.

Lemma 22 (Barb Preservation) *Assume* $\Gamma \vdash_{\mathrm{pc}} R$*,* $R{\downarrow_\alpha}$ *and* $|\,\alpha\,| \in \mathrm{low}(\Gamma)$*. Let* $i \in \{1, 2\}$*. Then,* $\mathrm{pc} \in L$ *and* $(\pi_i R){\downarrow_\alpha}$ *hold.*

Proof. By induction on the derivation of $R{\downarrow_\alpha}$. Let $x = |\,\alpha\,|$.

Case $R = M + x(\tilde{y}).P$. By T-SUB, T-NORMAL and T-RECV, the typing hypothesis $\Gamma \vdash_{\mathrm{pc}} R$ yields $\Gamma(x) \leq \langle \cdot \rangle^-_{\mathrm{pc}'}$, where $\mathrm{pc} \leq \mathrm{pc}'$. Thus, $\Gamma(x)$ must be of the form $\langle \cdot \rangle^{\cdot}_{\mathrm{pc}'}$. Because $x \in \mathrm{low}(\Gamma)$, this entails $\mathrm{pc}' \in L$. Because L is downward-closed, pc $\in L$ follows. Lastly, $\pi_i R$ is $\pi_i M + x(\tilde{y}).\pi_i P$, so $(\pi_i R)\downarrow_\alpha$ holds.

Case $R = M + \bar{x}\langle\tilde{z}\rangle.P$ is analogous.

Case $R = E[R_0]$, where $R_0\downarrow_\alpha$ and E does not bind x. Then, by T-PAR, T-NEW or T-BRACKET, together with T-SUB, $\Gamma \vdash_{\text{pc}} R$ implies $\Gamma_0 \vdash_{\text{pc}_0} R_0$, where Γ and Γ_0 agree on x and $\text{pc}_0 \geq \text{pc}$. The induction hypothesis then yields $(\pi_i R_0)\downarrow_\alpha$ and $\text{pc}_0 \in L$. Now, given $\text{pc}_0 \in L$ and recalling that L is downward-closed, the judgement $\Gamma \vdash_{\text{pc}_0} R_0$ cannot possibly be the premise of a derivation involving T-BRACKET. So, E must be standard. This implies $\pi_i R = \pi_i E[R_0] = E[\pi_i R_0]$, so $(\pi_i R)\downarrow_\alpha$ holds. Lastly, pc $\in L$ follows from $\text{pc}_0 \geq \text{pc}$ and $\text{pc}_0 \in L$. □

Then, it is easy to establish that any two processes which are respectively the left- and right-hand projections of a single, well-typed $\langle\pi\rangle$-calculus process are barbed bisimilar, provided only low-security barbs are observed. (We write $\Gamma \vdash R$ to indicate that $\Gamma \vdash_{\text{pc}} R$ holds for some pc $\in \mathcal{L}$.)

Lemma 23 (Barbed Bisimulation) *Let* $\mathbf{P}\ \mathcal{R}_\Gamma\ \mathbf{Q}$ *hold if and only if, for some* $\langle\pi\rangle$*-calculus process* R*, both* $\Gamma \vdash R$ *and* $\mathbf{P} \leq_0^\star \cdot \pi_1^{-1}\ R\ \pi_2 \cdot \geq_0^\star \mathbf{Q}$ *hold. Then,* $\mathcal{R}_\Gamma$ *is a weak* $\text{low}(\Gamma)$*-bisimulation.*

Proof. We prove that $\mathcal{R}_\Gamma$ is a weak $\text{low}(\Gamma)$-simulation. (To deal with ${\mathcal{R}_\Gamma}^{-1}$, simply swap π_1 and π_2.) First, assume $\mathbf{P}\ \mathcal{R}_\Gamma\ \mathbf{Q}$ and $\mathbf{P} \Rightarrow \mathbf{P}'$. Then, we have

$$\begin{tikzcd}[column sep=small]
\mathbf{P} \arrow[d, Rightarrow] & \leq_0^\star & \cdot & \pi_1^{-1} & R \arrow[d, Rightarrow] & \pi_2 & \cdot \arrow[d, Rightarrow] & \geq_0^\star & \mathbf{Q} \arrow[dll, Rightarrow, bend left] \\
\mathbf{P}' & \leq_0^\star & \cdot & \pi_1^{-1} & R' & \pi_2 & \mathbf{Q}' & &
\end{tikzcd}$$

The leftmost commutative diagram is given by Lemma 16, the middle one by Lemma 3, and the rightmost one merely stems from the fact that $\leq_0$ is contained within $\equiv$. Furthermore, Lemmas 18 and 21 yield $\Gamma \vdash R'$, which shows that $\mathbf{P}'\ \mathcal{R}_\Gamma\ \mathbf{Q}'$ holds.

Next, we check that $\mathcal{R}_\Gamma$ preserves *strong* barbs, which, given the above, implies that it also preserves weak ones, meeting the second condition of Definition 13. Assume $\mathbf{P}\ \mathcal{R}_\Gamma\ \mathbf{Q}$, $\mathbf{P}\downarrow_\alpha$ and $|\,\alpha\,| \in \text{low}(\Gamma)$. For some process R, $\Gamma \vdash R$ and $\mathbf{P} \leq_0^\star \cdot \pi_1^{-1}\ R\ \pi_2 \cdot \geq_0^\star \mathbf{Q}$ hold. As a result of the latter, we must have $R\downarrow_\alpha$. By Lemma 22, this implies $(\pi_2 R)\downarrow_\alpha$. Because $\pi_2 R \geq_0^\star \mathbf{Q}$ holds, $\mathbf{Q}\downarrow_\alpha$ follows. □

As a corollary, we show that, for any process R, the projections $\pi_1 R$ and $\pi_2 R$ are weakly barbed-congruent [17], provided we admit only contexts which, according to the type system, emit only low-security barbs.

Theorem 1 *For any process* R*, for any (standard) context* $\mathbf{C}$ *such that* $\Gamma \vdash \mathbf{C}[R]$ *holds in some* L*-environment* Γ*,* $\mathbf{C}[\pi_1 R]$ *and* $\mathbf{C}[\pi_2 R]$ *are weakly bisimilar.*

Proof. Apply Lemma 23. For $i \in \{1,2\}$, $\pi_i \mathbf{C}[R]$ is $\mathbf{C}[\pi_i R]$. As a result, $\mathbf{C}[\pi_1 R]$ and $\mathbf{C}[\pi_2 R]$ are related by $\mathcal{R}_\Gamma$. Thus, they are weakly $\text{low}(\Gamma)$-bisimilar. The result follows from $\text{low}(\Gamma) = \mathcal{N}$. □

Our second corollary is in the style of Honda *et al.*'s noninterference claim [11]:

Theorem 2 *Assume* $\Gamma_0 \vdash_{\text{pc}_0} \mathbf{P}_i$*, where* $\text{pc}_0 \notin L$*, holds for* $i \in \{1,2\}$*. Then, for any context* $\mathbf{C}$ *and for any environment* Γ *such that* $\Gamma \vdash \mathbf{C}[]$ *holds under the assumption* $\Gamma_0 \vdash_{\text{pc}_0} []$*,* $\mathbf{C}[\mathbf{P}_1]$ *and* $\mathbf{C}[\mathbf{P}_2]$ *are weakly* $\text{low}(\Gamma)$*-bisimilar.*

Proof. Because $\text{pc}_0 \notin L$, T-BRACKET yields $\Gamma_0 \vdash_{\text{pc}_0} \langle\mathbf{P}_i\rangle_i$ for $i \in \{1,2\}$. Define R as $\mathbf{C}[\langle\mathbf{P}_1\rangle_1 \mid \langle\mathbf{P}_2\rangle_2]$. According to T-PAR and to our hypothesis about $\mathbf{C}$, $\Gamma \vdash R$ holds. Apply Lemma 23. $\mathbf{C}[\mathbf{P}_1]$ and $\mathbf{C}[\mathbf{P}_2]$ are related by $\mathcal{R}_\Gamma$. As a result, they are weakly $\text{low}(\Gamma)$-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 $\mathbf{P}_1$ and $\mathbf{P}_2$ as entirely separate. However, Theorem 1 is expressed in terms of a nonstandard theoretical tool, namely the $\langle\pi\rangle$-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 subprocess with a security level $l \in \mathcal{L}$, indicating that its presence should not affect observers whose security clearance is not at least l. Thus, we introduce a "colored" π-calculus. It is reminiscent of Abadi, Lampson, and Lévy labelled λ-calculus [2] and of Sewell and Vitek's colored box-π-calculus [21]. However, we do not need to define a semantics for it; here, we view it only as a programming notation.

$$\begin{aligned} N &::= x(\tilde{x}).P \mid \bar{x}\langle\tilde{x}\rangle.P \mid \mathbf{0} \mid N + N \\ P &::= N \mid (P \mid P) \mid !P \mid \nu x.P \mid l : P \end{aligned}$$

The only variation with respect to the $\langle\pi\rangle$-calculus is that the construct $\langle\mathbf{P}\rangle_i$ 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

$$\text{T-COLOR}\quad \frac{\Gamma \Vdash_{\text{pc}} P \qquad l \leq \text{pc}}{\Gamma \Vdash_{\text{pc}} l : P}$$

Note that the new type system is no longer parameterized by a set L. We use $\Gamma \Vdash_{\text{pc}} P$ 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 $\lfloor\cdot\rfloor_L$, which drops all sub-terms whose color is not a member of L, and produces a term in the standard π-calculus. The function which strips off all colors, namely $\lfloor\cdot\rfloor_{\mathcal{L}}$, is written $\lfloor\cdot\rfloor$.

Definition 15 *Let $\lfloor\cdot\rfloor_L$ satisfy $\lfloor l : P\rfloor_L = \lfloor P\rfloor_L$ when $l \in L$, $\lfloor l : P\rfloor_L = \mathbf{0}$ when $l \notin L$, 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 π-calculus, for any (standard) context $\mathbf{C}$ such that $\Gamma \Vdash \mathbf{C}[R]$ holds in some L-environment Γ, $\mathbf{C}[\lfloor R\rfloor]$ and $\mathbf{C}[\lfloor R\rfloor_L]$ are weakly bisimilar.*

Proof. Define a mapping $\cdot^\star$ from the colored π-calculus into the $\langle\pi\rangle$-calculus, which satisfies $(l : P)^\star = P^\star$ when $l \in L$, $(l : P)^\star = \langle\lfloor P\rfloor\rangle_1$ when $l \notin L$, and is a homomorphism on standard process forms. Then, the identities $\pi_1(P^\star) = \lfloor P\rfloor$ and $\pi_2(P^\star) = \lfloor P\rfloor_L$ hold for any colored process P. Furthermore, it is easy to check that $\Gamma \Vdash_{\text{pc}} P$ implies $\Gamma \vdash_{\text{pc}} P^\star$. As a consequence, $\Gamma \Vdash \mathbf{C}[R]$ yields $\Gamma \vdash (\mathbf{C}[R])^\star$. Furthermore, $(\mathbf{C}[R])^\star$ is $\mathbf{C}[R^\star]$. By Theorem 1, $\mathbf{C}[\pi_1(R^\star)]$ and $\mathbf{C}[\pi_2(R^\star)]$ are weakly bisimilar. By the identities above, these are none other than $\mathbf{C}[\lfloor R\rfloor]$ and $\mathbf{C}[\lfloor R\rfloor_L]$. $\square$

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

7 Discussion

7.1 Asynchrony

Our type system allows input (resp. output) channel types $\langle\tilde{t}\rangle_l^-$ (resp. $\langle\tilde{t}\rangle_l^+$) to be covariant (resp. contravariant) in their *parameters* $\tilde{t}$, but both are *invariant* in their security level l (Definition 10). 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 π-calculus, where every sender is of the form $\bar{x}\langle\tilde{z}\rangle.\mathbf{0}$. 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 $\mathcal{L} = \{\text{L}, \text{H}\}$, with $\text{L} \leq \text{H}$. Take $L = \{\text{L}\}$. Consider the typing judgement

$$x : \langle\ \rangle_{\text{H}}^{\pm}, y : \langle\ \rangle_{\text{L}}^{\pm}, z : \langle\ \rangle_{\text{L}}^{+} \vdash_{\text{L}} \langle\bar{x}\rangle_2 \mid \bar{y} \mid x.y.\mathbf{0} \mid y.\bar{z}$$

This judgement is *incorrect* in our type system, because the sub-term $x.y.\mathbf{0}$ is ill-typed. Indeed, listening on channel x, which has type $\langle\ \rangle_{\text{H}}^{\pm}$, causes "pc" to become H. Subsequently, listening on channel y becomes illegal, since y has type $\langle\ \rangle_{\text{L}}^{\pm}$. On the other hand, if input channel types were covariant, then $\langle\ \rangle_{\text{L}}^{\pm}$ would be a subtype of $\langle\ \rangle_{\text{H}}^{-}$, and the judgement would become correct. Yet the two projections of this process, namely

$$\bar{y} \mid x.y.\mathbf{0} \mid y.\bar{z} \qquad \text{and} \qquad \bar{x} \mid \bar{y} \mid x.y.\mathbf{0} \mid y.\bar{z}$$

are *not* weakly $\{y, z\}$-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 $\bar{x}$ 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 π-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 $x.y.\mathbf{0}$ 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. [24]), so its success alone does not carry any information. This fact is pointed out and exploited by Honda *et al.* [11, 12, 26]. It is indeed crucial, in practice, to take advantage of it, because the π-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 [27] 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:

$$\textsc{Join}$$
$$\langle \bar{x}.\mathbf{P}\rangle_1 \mid \langle \bar{x}.\mathbf{Q}\rangle_2 \mapsto \bar{x}.(\langle \mathbf{P}\rangle_1 \mid \langle \mathbf{Q}\rangle_2) \qquad \text{if } x \text{ is linear}$$

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 $\bar{x}$ is sent from a "high"-level process. The anticipated difficulty is in establishing the completeness lemma, i.e. the analogue of Lemma 16. 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 [19], 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-π-calculus [21].) We suggested, at the time, that this approach should be applicable to the π-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* π-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 [21] 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 (Deg_s) with T-SUB), and nonlinear channel types are invariant in their security level. The noninterference results stated in [11, 12, 26] 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 [22, 23, 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

[1] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Conference Record of the 26th ACM Symposium on Principles of Programming Languages*, pages 147–160, San Antonio, Texas, Jan. 1999. ACM Press. URL: `http://www.soe.ucsc.edu/~abadi/Papers/flowpopl.ps`.

[2] M. Abadi, B. Lampson, and J.-J. Lévy. Analysis and caching of dependencies. In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming*, pages 83–91, Philadelphia, Pennsylvania, May 1996.

ACM Press. URL: http://www.soe.ucsc.edu/~abadi/Papers/make-preprint.ps.

[3] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, The MITRE Corp., Bedford, Massachusetts, July 1975. URL: http://www.mitre.org/resources/centers/infosec/infosec.html.

[4] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis of processes for no read-up and no write-down. In W. Thomas, editor, *Proceedings of FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 120–134. Springer, Mar. 1999. URL: http://www.di.unipi.it/~chiara/publ-40/BDNN99.ps.

[5] G. Boudol and I. Castellani. Non-interference for concurrent programs and thread systems. To appear. URL: ftp://ftp-sop.inria.fr/mimosa/personnel/gbo/non-interf-threads.ps.gz, Sept. 2001.

[6] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Massachusetts, 1982.

[7] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995. URL: http://www.cs.unibo.it/~gorrieri/Papers/jcsfinal.ps.gz.

[8] M. Hennessy. The security picalculus and non-interference. Technical Report 2000:05, University of Sussex, Nov. 2000. URL: ftp://ftp.cogs.susx.ac.uk/pub/reports/compsci/cs052000.ps.Z.

[9] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science. Springer-Verlag, July 2000. URL: http://www.depaul.edu/~jriely/papers/00icalp.ps.gz.

[10] M. Hepburn and D. Wright. Trust in the pi-calculus. In *Third International Conference on Principles and Practice of Declarative Programming (PPDP'01)*, Sept. 2001.

[11] K. Honda, V. Vasconcelos, and N. Yoshida. Secure information flow as typed process behaviour. In G. Smolka, editor, *Proceedings of the 2000 European Symposium on Programming (ESOP'00)*, volume 1782 of *Lecture Notes in Computer Science*, pages 180–199. Springer Verlag, Mar. 2000. URL: ftp://ftp.dcs.qmw.ac.uk/lfp/kohei/siftp-esop00.ps.gz.

[12] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL'02)*, pages 81–92, Portland, Oregon, Jan. 2002. URL: http://www.mcs.le.ac.uk/~nyoshida/paper/ifa1.ps.gz.

[13] A. Igarashi and N. Kobayashi. Type reconstruction for linear π-calculus with I/O subtyping. *Information & Computation*, 161:1–44, Aug. 2000. URL: http://www.graco.c.u-tokyo.ac.jp/~igarashi/papers/LinearPi.ps.gz.

[14] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the Pi-Calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, Sept. 1999. URL: http://www.acm.org/pubs/citations/journals/toplas/1999-21-5/p914-kobayashi/.

[15] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*. IEEE Press, 1994. URL: http://chacs.nrl.navy.mil/publications/CHACS/1994/1994mclean-sp.ps.

[16] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, Oct. 1991. URL: ftp://ftp.cl.cam.ac.uk/users/rm135/ppi.ps.Z.

[17] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695, Vienna, Austria, July 1992. Springer-Verlag. URL: ftp://ftp-sop.inria.fr/meije/theorie-par/davides/bn.ps.gz.

[18] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 376–385, June 1993. URL: http://www.cis.upenn.edu/~bcpierce/papers/pi-lics.ps.

[19] F. Pottier and S. Conchon. Information flow inference for free. In *Proceedings of the the 5th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 46–57, Montréal, Canada, Sept. 2000. ACM Press. URL: http://pauillac.inria.fr/~fpottier/publis/fpottier-conchon-icfp00.ps.gz.

[20] F. Pottier and V. Simonet. Information flow inference for ML. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL'02)*, pages 319–330, Portland, Oregon, Jan. 2002. ACM Press. URL: http://pauillac.inria.fr/~fpottier/publis/fpottier-simonet-popl02.ps.gz.

[21] P. Sewell and J. Vitek. Secure composition of untrusted code: Wrappers and causality types. In *Proceedings of the 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000. URL: http://www.cl.cam.ac.uk/users/pes20/wraptypes.ps.gz.

[22] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of the 25th ACM Symposium on Principles of Programming Languages*, pages 355–364, Jan. 1998. URL: http://www.cs.nps.navy.mil/people/faculty/volpano/papers/popl98.ps.Z.

[23] G. S. Smith. A new type system for secure information flow. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 115–125, Cape Breton, Nova Scotia, June 2001. URL: http://www.cs.fiu.edu/~smithg/papers/csfw01.ps.gz.

[24] N. Yoshida. Graph types for monadic mobile processes. In V. Chandru and V. Vinay, editors, *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer-Verlag, 1996. URL: http://www.mcs.le.ac.uk/~nyoshida/paper/graph1_short.ps.gz.

[25] N. Yoshida, K. Honda, and M. Berger. Linearity and bisimulation. Technical Report MSC-2001/48, University of Leicester, Dec. 2001. URL: http://www.mcs.le.ac.uk/~nyoshida/paper/lb.ps.gz.

[26] N. Yoshida, K. Honda, and M. Berger. Linearity and bisimulation. In *Proceedings of 5th International Conference of Foundations of Software Science and Computer Structures (FoSSaCs 2002)*, Lecture Notes in Computer Science. Springer Verlag, Apr. 2002. URL: http://www.mcs.le.ac.uk/~nyoshida/paper/fossacs_ca_final.ps.gz.

[27] S. Zdancewic and A. C. Myers. Secure information flow and CPS. In D. Sands, editor, *Proceedings of the 2001 European Symposium on Programming (ESOP'01)*, Lecture Notes in Computer Science, Genova, Italy, Apr. 2001. Springer Verlag. URL: http://www.cs.cornell.edu/zdance/lincont.ps.