URL: http://www.elsevier.nl/locate/entcs/volume85.html 17 pages

Coordination Model and Noninterference

Alessandro Aldini [1,2]

Istituto di Scienze e Tecnologie dell'Informazione
Università di Urbino "Carlo Bo"
Urbino, Italy

Abstract

Noninterference properties for the analysis of secure information flow are proposed in the setting of a process algebra modeling some Linda coordination primitives (asynchronous communication and read operation). To this end, relaxed definitions of equivalence are introduced that take into consideration the observational power of the external observer. The resulting taxonomy is compared with corresponding security definitions for synchronous communication models. As a result, we emphasize how the proposed coordination model affects the expressive power of some noninterference properties, by giving a new intuition to the relative merits.

1 Introduction

Confidentiality issues in multi-level security systems have been deeply and successfully treated in several formal models. In particular, we want to focus on those semantics-based models (i.e., models that analyze the program behavior to verify security properties) that rely on the noninterference approach to information flow analysis (see, e.g., [5] and the references therein). The basic idea of noninterference [7] is that lack of implicit information flow from high- to low-security levels is ensured if the interactions observed at the low-security level are invariant under changes in high-level behaviors. In practice, by interacting with the low-level interface of the system at hand, an observer cannot deduce anything about the interplay between a high-class user and the high-security part of the system. A common feature of most of the noninterference-based formal models is the synchrony of both input and output operations, which represent the possible interactions between the system and the environment. In essence, the main principle behind a synchronous communication model is that the environment and the modeled system must

[1] This work has been supported by MEFISTO project "Metodi Formali per la Sicurezza e il Tempo".
[2] Email: aldini@sti.uniurb.it

agree on the events to be performed if we want that they really happen. For instance, a confidential high-level output takes place the instant a high-class user of the environment is available to accept it. Symmetrically, an input operation is allowed to be performed the instant the environment activates that input. On the basis of such a symmetric treatment of communication primitives, the system at hand is analyzed in order to establish whether or not a low-security observer can distinguish alternative behaviors of the system that differ in the high-level operations only. The motivation for symmetrically treating inputs and outputs in security issues is argued in many works (see, e.g., [8]). Anyway, in many cases a model that does not distinguish output actions from input ones can lead to noninterference results that turn out to be too restrictive. In fact, there may be several output communications that cannot be refused, such as the appearance of information on a screen, and that allow the system to proceed just after performing them (as emphasized, e.g., in [13]). Since their occurrence is entirely uninfluenced by the high-level environment behavior, confidential output operations cannot cause an information leakage from a high-level user to a low-level unauthorized observer that interact with the same system. In essence, the execution of an asynchronous high-security output can be neither refused (delayed) by the high-level user nor observed by the low-level user.

In this paper, we analyze the effect of considering output events as asynchronous communications on the expressive power of some noninterference properties. In particular, we consider the CCS-based security properties introduced in [4] to describe in a process algebraic setting the basic idea of noninterference. The formal framework we employ, which is borrowed from [3], expresses some Linda coordination primitives (asynchronous communication and read operation) in the setting of a nondeterministic process algebra inspired by CCS [12]. In this model, asynchronous communication is realized by means of a so-called *tuplespace*, which models a shared box that explicitly (i) receives messages (also called tuples) generated via output operations, and (ii) allows processes of the environment to remove (read without removing) the same messages via input (read) operations. Hence, from the observability viewpoint, the focus moves from the behavior of the system, which can interact with the environment via the execution of inputs and outputs, to the set of tuples that can be removed (read) from the tuplespace. In other words, since communications take place through access to the tuplespace, what an observer can see is just the tuplespace. As a consequence, we have that a read operation performed by the system has the same effect as an internal action, since it does not cause changes in the tuplespace. Obviously, this requires that the equivalence relations used to define the observational semantics of our language must take into consideration the observational power of an external user. To this aim, two classical notions of equivalence will be rephrased in this asynchronous communication model, i.e. a trace equivalence and a weak bisimulation equivalence.

As far as the security context is concerned, the tuplespace is divided into a high-level part containing high-security tuples, which are emitted (and can be consumed) by high-level users, and a low-level part containing unclassified, public messages, which are generated (and can be consumed) by low-level users. To make it clear the relationship between such an asynchronous communication model and the related security issues, some noninterference properties are defined in the same line of [4]. One goal consists of studying the kind of information flow that can be revealed in a Linda-like coordination model, where users deal with an explicit communication interface, the tuplespace, instead of directly interacting with the system, and a low-level observer can infer the behavior of high-level users by manipulating the public portion of the tuplespace only. On the other hand, another result we present is a comparison of the resulting taxonomy with the classification of the corresponding properties of [4]. The emphasis is on the influence of the described coordination model upon the security analysis of concurrent computer systems.

The rest of the paper is organized as follows. In Sect. 2 we introduce the process algebra and the formal framework necessary to conduct the noninterference analysis. Then, in Sect. 3 we propose in this asynchronous setting some security properties of [4] and, through several examples, we study their expressive power, by emphasizing similarities and differences with respect to the synchronous setting of [4]. Finally, in Sect. 4 we report on related work and some conclusions.

2 Security Asynchronous Language

With respect to synchronous languages, where the constituent elements are actions representing system activities, the basic elements of the asynchronous calculus are messages, which can be put in (removed from) the tuplespace. Formally, we denote by $\mathcal{M}$, ranged over by $a, b, \ldots$, the set of message names. As usual in security models, we distinguish between high-level message names, denoted by set $\mathcal{M}_H$, and low-level message names, denoted by set $\mathcal{M}_L$, so that $\mathcal{M}_H$ and $\mathcal{M}_L$ are two disjoint sets that form a covering of $\mathcal{M}$.

The security asynchronous language, here called *SAL*, is a slight variant of LINPA [3], which in turn is an asynchronous version of CCS [12]. In particular, *SAL* is equipped with the input, output, and read operations, and enriched with the hiding operator, which is needed for the definition of security properties. Indeed, abstraction is used to specify the observational power of each external observer, depending on the security level of such an observer. The set of *process* terms is generated by the grammar:

$$C ::= \underline{0} \mid \mu.C \mid C|C \mid C + C \mid C\backslash a \mid C/a \mid Z$$

where $\underline{0}$ is the empty process (we usually omit it when it is clear from the context), and $\_|\_, \_ + \_, \_\backslash a, \_/a$ denote the usual parallel, choice, restriction, and hiding operators. Moreover, for each constant Z we have a corresponding

definition $Z \stackrel{def}{=} C$, where C is guarded on constants [12]. The possible prefixes μ are:

$$\mu ::= \tau \mid in(a) \mid out(a) \mid rd(a)$$

where τ is the internal, unobservable action, and $in(a)$ and $out(a)$ express the usual input and output primitives of Linda, respectively. More precisely, $out(a)$ produces, in one internal step, a new tuple $\langle a \rangle$ (containing message a) that is put in the tuplespace. Hence, an output operation is non-blocking as it allows the system to proceed just after performing the rendering of the tuple. Instead, $in(a)$ removes, if present, the tuple $\langle a \rangle$ from the tuplespace. Note that if such a tuple is not present in the tuplespace, then the input operation is blocked. Similarly, $rd(a)$ denotes the blocking reading of any message a without removing it from the tuplespace. Tuples are not considered in the syntax of processes; instead, they are described as *states*, which are defined as terms generated by the syntax:

$$P ::= \langle a \rangle \mid C \mid P|P \mid P\backslash a \mid P/a.$$

In practice, states model the parallel composition of tuples that are present in the tuplespace and processes that handle the tuplespace. We call $\mathcal{A}$ the set of possible agents generated by the grammar above, ranged over by $P, Q, \ldots$.

The operational semantics of SAL is defined through the structural congruence $\equiv$, which is defined as the smallest congruence that satisfies the axioms of Table 1. In axioms (x) and (xi), function $fn(P)$ denotes the set of free names of P and is defined as follows:

$$fn(\underline{0}) = \emptyset$$

$$fn(\langle a \rangle) = \{a\}$$

$$fn(P|Q) = fn(P + Q) = fn(P) \cup fn(Q)$$

$$fn(P\backslash a) = fn(P/a) = fn(P)\backslash\{a\}$$

$$fn(in(a).P) = fn(out(a).P) = fn(rd(a).P) = \{a\} \cup fn(P)$$

$$fn(Z) = fn(P) \text{ if } Z \stackrel{def}{=} P.$$

With abuse of notation, in axiom (xii), which is α–conversion, we use $P[b/a]$ to denote the term obtained by renaming all the free occurrences of the name a in P with the fresh name b. Then, we define the operational semantics of SAL as the labelled transition system $(\mathcal{A}, Act, \rightarrow)$, where (i) the states are agents of the language; (ii) Act is a set of transition labels (ranged over by $\pi, \pi', \ldots$) defined as $Act = \{\tau\} \cup O \cup I \cup R$, where $O = \{\bar{a} \mid a \in \mathcal{M}\}$ is the set of labels expressing offer of tuples of the tuplespace to the environment, $I = \mathcal{M}$ is the set of labels denoting consumption of tuples from the tuplespace, and $R = \{\underline{a} \mid a \in \mathcal{M}\}$ is the set of labels denoting reading of tuples of the tuplespace; (iii) the transition relation $\rightarrow \subseteq \mathcal{A} \times Act \times \mathcal{A}$ is defined as the least relation satisfying the axioms and the rules in Table 2. In the following,

(i)	$P\|\underline{0} \equiv P$	
(ii)	$P\|Q \equiv Q\|P$	
(iii)	$(P\|Q)\|R \equiv P\|(Q\|R)$	
(iv)	$P + \underline{0} \equiv P$	
(v)	$P + P \equiv P$	
(vi)	$P + Q \equiv Q + P$	
(vii)	$(P + Q) + R \equiv P + (Q + R)$	
$(viii)$	$\underline{0}\, op\, a \equiv \underline{0}$	
(ix)	$(P op\, a) op\, b \equiv (P op\, b) op\, a$	
(x)	$(P + Q) op\, a \equiv P + (Q op\, a)$	$a \notin fn(P)$
(xi)	$(P\|Q) op\, a \equiv P\|(Q op\, a)$	$a \notin fn(P)$
(xii)	$P op\, a \equiv P[b/a] op\, b$	b fresh
$(xiii)$	$Z \equiv P$	$Z \stackrel{def}{=} P$
	$op \in \{\backslash, /\}$	

Table 1
Structural congruence for SAL

we assume that set $\mathcal{L} = O \cup I \cup R$ is ranged over by $\alpha, \alpha', \ldots$. Since the set of message names is partitioned into high-level names and low-level ones, we also partition $\mathcal{L}$ into high-level labels and low-level ones, denoted by sets Act_H and Act_L, respectively, such that $\alpha \in \{a, \bar{a}, \underline{a}\}$ belongs to Act_H (Act_L) iff $a \in \mathcal{M}_H$ ($a \in \mathcal{M}_L$).

When defining security properties, it will be useful to employ a restriction operator $P\backslash_I a$, which limits its scope to input transition labels only, whose semantics is defined as follows:

$$\frac{P \stackrel{\pi}{\rightarrow} P'}{P\backslash_I a \stackrel{\pi}{\rightarrow} P'\backslash_I a} \;\; \pi \neq a.$$

Similarly, it is possible to define the operator $P\backslash_O a$, which prevents the execution of $\bar{a}$ labelled transitions only, and, in the same line, the operator $P/_O a$, which hides $\bar{a}$ labelled transitions only.

$$(prefix)\quad \tau.P \xrightarrow{\tau} P \qquad out(a).P \xrightarrow{\tau} \langle a\rangle|P$$

$$in(a).P \xrightarrow{a} P \qquad rd(a).P \xrightarrow{\underline{a}} P$$

$$(tuple)\quad \langle a\rangle \xrightarrow{\bar{a}} \underline{0}$$

$$(sum)\quad \frac{P \xrightarrow{\pi} P'}{P+Q \xrightarrow{\pi} P'}$$

$$(par)\quad \frac{P \xrightarrow{\pi} P'}{P|Q \xrightarrow{\pi} P'|Q}$$

$$\frac{P \xrightarrow{a} P' \;\; Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad \frac{P \xrightarrow{\underline{a}} P' \;\; Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q}$$

$$(res)\quad \frac{P \xrightarrow{\pi} P'}{P\backslash a \xrightarrow{\pi} P'\backslash a} \;\; \pi \notin \{a, \bar{a}, \underline{a}\}$$

$$(hid)\quad \frac{P \xrightarrow{\pi} P'}{P/a \xrightarrow{\tau} P'/a} \;\; \pi \in \{a, \bar{a}, \underline{a}\} \qquad \frac{P \xrightarrow{\pi} P'}{P/a \xrightarrow{\pi} P'/a} \;\; \pi \notin \{a, \bar{a}, \underline{a}\}$$

$$(congr)\quad \frac{P \equiv Q \;\; Q \xrightarrow{\pi} Q' \;\; P' \equiv Q'}{P \xrightarrow{\pi} P'}$$

Table 2
Operational semantics of *SAL*

2.1 Equivalences

Since the analysis of security properties is based on equivalence checking and focuses on the observable interactions between the environment and the system at hand, we need an equivalence relation that abstracts away from invisible τ actions. Moreover, in the setting of an asynchronous language, we also must pay attention on what the external observer is allowed to see. As emphasized in [3], an observer can see and manipulate the tuplespace, while he has no means for inferring whether or not the system is executing input (read) operations (this intuition leads, e.g., to the definition of a barbed bisimulation [1]). Based on these considerations, we define two adequate equivalences for *SAL*, i.e. a trace equivalence and a weak bisimulation equivalence (which extends the strong *rd*-bisimulation of [3]), which can be used to express security properties. In essence, we join the requirements expressed in [3], which take into consideration the observational power of the external user, with the usual definitions employed in [4,6].

Definition 2.1 The expression $P \stackrel{\alpha}{\Rightarrow} P'$ stands for $P(\stackrel{\tau}{\rightarrow})^* P_1 \stackrel{\alpha}{\rightarrow} P_2(\stackrel{\tau}{\rightarrow})^* P'$, where $(\stackrel{\tau}{\rightarrow})^*$ denotes a possibly empty sequence of τ labelled transitions. Denoted with $\gamma = \alpha_1, \ldots, \alpha_n \in \mathcal{L}^*$ a sequence of visible actions, the expression $P \stackrel{\gamma}{\Rightarrow} P'$ stands for $P \stackrel{\alpha_1}{\Rightarrow} P_1 \stackrel{\alpha_2}{\Rightarrow} \ldots \stackrel{\alpha_{n-1}}{\Rightarrow} P_{n-1} \stackrel{\alpha_n}{\Rightarrow} P'$. If γ is the empty sequence $\langle\rangle$, then $P \stackrel{\langle\rangle}{\Rightarrow} P'$ stands for $P(\stackrel{\tau}{\rightarrow})^* P'$. We say that P' is reachable from P, denoted $P \Rightarrow P'$, if $\exists \gamma : P \stackrel{\gamma}{\Rightarrow} P'$.

With abuse of notation, we say that $\alpha \in \gamma = \alpha_1, \ldots, \alpha_n$ if $\alpha = \alpha_i$ for some $1 \leq i \leq n$. We define the set $\underline{A}_\gamma$ of reading operations occurring in γ as $\underline{A}_\gamma = \{\alpha \in \gamma \mid \exists a \in \mathcal{M} : \alpha = \underline{a}\}$. Given a sequence γ, where $\underline{A}_\gamma = \{\underline{a}_1, \ldots, \underline{a}_m\}$, we define S_γ as the set of sequences obtained from γ by replacing each $\underline{a}_i$ $(1 \leq i \leq m)$ within γ with $(\underline{a}_i)^*$. Note that $\gamma \in S_\gamma$ and $S_\gamma = \{\gamma\}$ if $\underline{A}_\gamma = \emptyset$.

The trace equivalence for *SAL* is defined as follows:

Definition 2.2 Given $P \in \mathcal{A}$, the set $T(P)$ of traces associated with P is defined as $T(P) = \{\gamma \in \mathcal{L}^* \mid \exists P' : P \stackrel{\gamma}{\Rightarrow} P'\}$.

Given $P, Q \in \mathcal{A}$, we say that P and Q are *rd*-trace equivalent, denoted $P \approx_{\mathrm{T}_{rd}} Q$, if and only if for each $\gamma \in T(P)$ there exists $\delta \in T(Q)$ such that $\delta \in S(\gamma)$, and *vice versa*.

In the following, the expression $P \stackrel{\hat{\pi}}{\Rightarrow} P'$ stands for $P \stackrel{\pi}{\Rightarrow} P'$ if $\pi \neq \tau$, and for $P(\stackrel{\tau}{\rightarrow})^* P'$ if $\pi = \tau$. Note that $(\stackrel{\tau}{\rightarrow})^*$ stands for zero or more internal transitions, while $\stackrel{\tau}{\Rightarrow}$ requires at least one internal transition.

The weak bisimulation equivalence for *SAL* is defined as follows:

Definition 2.3 An equivalence relation $\mathcal{R}$ on $\mathcal{A}$ is a weak *rd*-bisimulation if $(P, Q) \in \mathcal{R}$ implies:

- if $P \stackrel{\pi}{\rightarrow} P'$, with $\pi \neq \underline{a}$, then $\exists Q'$ such that $Q \stackrel{\hat{\pi}}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{R}$;
- if $P \stackrel{\underline{a}}{\Rightarrow} P'$ then $\exists Q'$ such that $Q((\stackrel{\tau}{\rightarrow})^*(\stackrel{\underline{a}}{\rightarrow})^*)^* Q'$ and $(P', Q') \in \mathcal{R}$.

Two agents P and Q are weakly *rd*-bisimilar, written $P \approx_{\mathrm{B}_{rd}} Q$, if there exists a weak *rd*-bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

Based on the definitions above, an $\underline{a}$ labelled transition can be simulated with a mixed (possibly empty) sequence of τ and $\underline{a}$ labelled transitions. The intuition, which is the weak version of the idea reported in [3], is that since a read operation does not alter the content of the tuplespace, from the external observer standpoint the behavior of $rd(a)$ is the same as that expressed by a possibly empty sequence of unobservable actions and read operations. On the other hand, a τ labelled transition cannot be simulated with a mixed sequence of τ and $\underline{a}$ labelled transitions. This is because the behavior of $rd(a)$ is more restrictive than the behavior of an internal τ action, since the execution of $rd(a)$ implies that a tuple $\langle a \rangle$ is in the tuplespace.

3 Noninterference Properties

In this section, we formalize some noninterference properties based on the two equivalence relations introduced in the previous section. The definitions we present are based on the classification of security properties described in [4,6] for a synchronous communication model. In the setting of an asynchronous communication model, the goal of such definitions is to verify whether or not a low-level user can infer the behavior of the high-level user by observing the tuplespace during the agent execution.

For the sake of readability, we assume that if $L = \{a_1, \ldots, a_n\} \subseteq \mathcal{M}$, then the abbreviation $P\ op\ L$ stands for $P\ op\ a_1 \ldots op\ a_n$, with $op \in \{\backslash, /\}$.

3.1 Trace-based Properties

The definition of noninterference states that there is no observable distinction (by the low-level user) between the behavior of the system when it accepts high-level inputs and the behavior of the same system when it does not interact with the high-level environment. In the setting of *SAL*, this means that the low-level content of the tuplespace is not affected by high-level input operations. More precisely, a low-level user should not be able to guess the content of the high-level portion of the tuplespace by interacting with the low-level part of the tuplespace. We rephrase the related property, called Nondeterministic Noninterference (NNI) [4], in the context of *SAL*, by defining the *rd*-trace based NNI (*rd*-NNI).

Definition 3.1 (*rd*-NNI) $P \in rd\text{-NNI} \Leftrightarrow P/\mathcal{M}_H \approx_{\mathrm{T}_{rd}} (P\backslash_I \mathcal{M}_H)/\mathcal{M}_H$.

In synchronous communication models, NNI is not adequate to reveal interferences due to high-level outputs. Therefore, it is necessary to strengthen such a property by symmetrically treating input actions and output actions. This leads to the definition of Strong Nondeterministic Noninterference (SNNI), which states that both high-level inputs and high-level outputs should not affect the low-level behavior of the system. However, in an asynchronous setting, such an extension is not needed, as we now formally show in the context of *SAL*. In particular, since output is asynchronous, the high-level user cannot prevent the system from executing an $out(a)$ operation or, in other words, there is no relation between the emission of a message performed by the system and the behavior of the high-level user. As a consequence, such a kind of operation cannot help the low-level user to deduce the behavior of the high-level user.

Definition 3.2 (*rd*-SNNI) $P \in rd\text{-SNNI} \Leftrightarrow P/\mathcal{M}_H \approx_{\mathrm{T}_{rd}} P\backslash\mathcal{M}_H$.

In order to prove the proposition stating that *rd*-NNI and *rd*-SNNI are equivalent in the context of *SAL*, we need the following lemma. As far as the notation is concerned, we employ the function $low : \mathcal{L}^* \to Act_L^*$, which takes

a trace γ and removes all the high-level actions from it (i.e., returns the low-level subsequence of γ), and the function $highinput : \mathcal{L}^* \rightarrow (Act_H \cap I)^*$, which extracts from a trace the subsequence composed of all its high-level inputs.

Lemma 3.3 *The following hold:*

(*i*) $P \in rd\text{-NNI} \Leftrightarrow \forall \gamma \in T(P), \exists \delta \in T(P)$ such that $low(\delta) \in S(low(\gamma)) \wedge highinput(\delta) = \langle\rangle$.

(*ii*) $P \in rd\text{-SNNI} \Leftrightarrow \forall \gamma \in T(P), \exists \delta \in T(P)$ such that $\delta \in S(low(\gamma))$.

Proof.

(*i*)

($\Rightarrow$) $\forall \gamma \in T(P)$, $low(\gamma) \in T(P/\mathcal{M}_H)$. Then, by hypothesis $\exists \gamma' \in T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H)$ such that $\gamma' \in S(low(\gamma))$ and $highinput(\gamma') = \langle\rangle$. The result follows from the fact that $\exists \delta \in T(P)$ such that $\gamma' = low(\delta)$ and $highinput(\delta) = \langle\rangle$.

($\Leftarrow$) It is trivial to see that $\forall \gamma \in T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H), \exists \delta \in T(P/\mathcal{M}_H)$ such that $\delta \in S(\gamma)$, since $T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H) \subseteq T(P/\mathcal{M}_H)$. On the other hand, $\forall \gamma \in T(P/\mathcal{M}_H), \exists \gamma' \in T(P)$ such that $low(\gamma') = \gamma$. By hypothesis, $\exists \delta \in T(P)$ such that $low(\delta) \in S(low(\gamma'))$ and $highinput(\delta) = \langle\rangle$. Therefore, it follows $\delta \in T(P\backslash_I \mathcal{M}_H)$ and $low(\delta) \in T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H)$. The result derives from $low(\delta) \in S(low(\gamma')) = S(\gamma)$.

(*ii*)

($\Rightarrow$) $\forall \gamma \in T(P)$, $low(\gamma) \in T(P/\mathcal{M}_H)$. Then, by hypothesis $\exists \delta \in T(P\backslash \mathcal{M}_H)$ such that $\delta \in S(low(\gamma))$, from which we immediately derive the result, since $\delta \in T(P)$.

($\Leftarrow$) It is trivial to see that $\forall \gamma \in T(P\backslash \mathcal{M}_H), \exists \delta \in T(P/\mathcal{M}_H)$ such that $\delta \in S(\gamma)$, since $T(P\backslash \mathcal{M}_H) \subseteq T(P/\mathcal{M}_H)$. On the other hand, $\forall \gamma \in T(P/\mathcal{M}_H)$, $\exists \gamma' \in T(P)$ such that $low(\gamma') = \gamma$. By hypothesis, $\exists \delta \in T(P)$ such that $\delta \in S(low(\gamma'))$, from which we derive the result, since $\delta \in T(P\backslash \mathcal{M}_H)$. □

Proposition 3.4 $P \in rd\text{-NNI} \Leftrightarrow P \in rd\text{-SNNI}$.

Proof.

($\Leftarrow$) We start by observing that $T(P\backslash \mathcal{M}_H) \subseteq T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H)$ and $T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H) \subseteq T(P/\mathcal{M}_H)$. By hypothesis, if $\gamma \in T(P/\mathcal{M}_H)$, then $\exists \delta \in T(P\backslash \mathcal{M}_H)$ (and, as a consequence, $\delta \in T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H)$) such that $\delta \in S(\gamma)$. On the other hand, if $\gamma \in T((P\backslash_I \mathcal{M}_H)/\mathcal{M}_H)$, then $\gamma \in T(P/\mathcal{M}_H)$ and, by hypothesis, $\exists \delta \in T(P\backslash \mathcal{M}_H)$ (and, as a consequence, $\delta \in T(P/\mathcal{M}_H)$) such that $\delta \in S(\gamma)$.

($\Rightarrow$) By hypothesis and by Lemma 3.3(*i*), $\forall \gamma \in T(P), \exists \delta \in T(P)$ such that $highinput(\delta) = \langle\rangle$ and $low(\delta) \in S(low(\gamma))$. Since $highinput(\delta) = \langle\rangle$ and since output is non-blocking, there exists $\delta' \in T(P)$ such that $\delta' = low(\delta)$. Hence, by Lemma 3.3(*ii*), we derive the result. □

To make it clear the intuition behind the asynchronous communication assumption, we propose a comparison with the synchronous case through some examples.

Example 3.5 Consider the agent $in(h).out(l).\underline{0}$, with $h \in \mathcal{M}_H$ and $l \in \mathcal{M}_L$. A low-security observer can easily distinguish between the situation in which the high-level user does not interact with the agent and that in which the agent consumes a tuple $\langle h \rangle$ previously emitted by the high-level user. Indeed, by verifying the presence of the tuple $\langle l \rangle$ in the tuplespace, a low-level observer can infer the high-level user behavior, even if the presence of $\langle h \rangle$ cannot be directly checked by the low-level user. Formally, we have that the agent is not *rd*-NNI secure (see Fig. 1). Similarly, in the synchronous setting of [4], the corresponding agent is not NNI secure, as the low-level output is blocked until the execution of the low-level input, which is entirely guided by the high-security user. Such an example shows that NNI and *rd*-NNI capture the same kind of interferences. This is because the synchronous and asynchronous models of communication consider inputs as blocking operations.

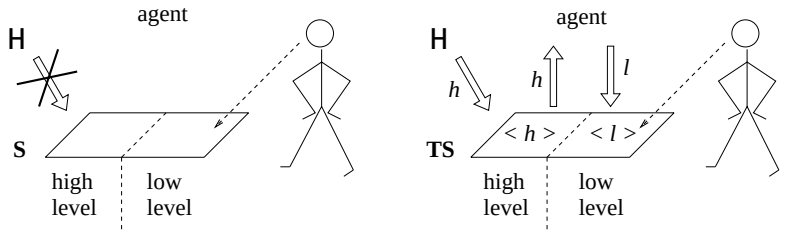

Fig. 1. Agent $in(h).out(l).\underline{0}$ – the tuplespace from the low-level viewpoint.

Now, replace the input operation in the agent above by a corresponding output operation, thus obtaining the new agent $out(h).out(l).\underline{0}$. By observing the low-security part of the tuplespace, the low-level user cannot deduce anything about the behavior of the high-level user. Indeed, the presence of the tuple $\langle l \rangle$ is not enough to establish whether or not the emitted tuple $\langle h \rangle$ has been consumed by the high-level user. Formally, the agent is *rd*-SNNI secure (see Fig. 2). On the contrary, in the synchronous setting of [4], the corresponding agent is NNI secure, but not SNNI secure, as the output operation is blocking, i.e. the execution of the low-level output reveals that a high-class user accepted the high-level output.

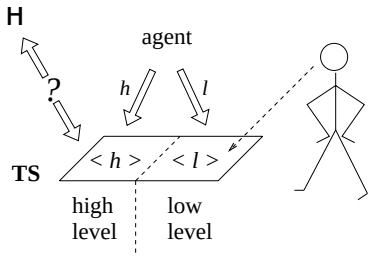

Fig. 2. Agent $out(h).out(l).\underline{0}$ – the tuplespace from the low-level viewpoint.

3.2 Bisimulation-based Properties

Similarly as seen in the previous section, we can define the noninterference property and its strong version on the basis of the weak *rd*-bisimulation equivalence defined in Sect. 2, thus obtaining the *rd*-Bisimulation NNI (*rd*-BNNI) and the *rd*-Bisimulation SNNI (*rd*-BSNNI).

Definition 3.6 (*rd*-BNNI, *rd*-BSNNI)

- $P \in$ *rd*-BNNI $\Leftrightarrow P/\mathcal{M}_H \approx_{\mathrm{B}_{rd}} (P\backslash_I \mathcal{M}_H)/\mathcal{M}_H$.
- $P \in$ *rd*-BSNNI $\Leftrightarrow P/\mathcal{M}_H \approx_{\mathrm{B}_{rd}} P\backslash\mathcal{M}_H$.

We first observe that, as it is easy to verify, $P \approx_{\mathrm{B}_{rd}} Q$ implies $P \approx_{\mathrm{T}_{rd}} Q$. Indeed, every trace of Q must be a trace also for P since P can simulate the behavior of Q, and *vice versa*. Hence, we have that *rd*-BNNI $\subseteq$ *rd*-NNI and *rd*-BSNNI $\subseteq$ *rd*-SNNI. However, with respect to the trace-based scenario, the equivalence relation between *rd*-BNNI and *rd*-BSNNI does not hold. This is because the weak *rd*-bisimulation equivalence is able to detect deadlocks and to discriminate agents also according to the nondeterministic structure of their labelled transition systems.

Example 3.7 Let us consider agent $P \stackrel{def}{=} out(h).in(h).out(l)$ (with $h \in \mathcal{M}_H$ and $l \in \mathcal{M}_L$) and verify the *rd*-BSNNI property. On the one hand, the semantics of $P\backslash\mathcal{M}_H$ is given by $P\backslash\mathcal{M}_H \stackrel{\tau}{\rightarrow} (\langle h\rangle \mid in(h).out(l))\backslash\mathcal{M}_H \stackrel{\tau}{\rightarrow} out(l)\backslash\mathcal{M}_H \stackrel{\tau}{\rightarrow} \langle l\rangle\backslash\mathcal{M}_H \stackrel{\bar{l}}{\rightarrow} \underline{0}$. Intuitively, if the high-level environment does not interact with the tuplespace, then the low-level user can read/consume the tuple $\langle l\rangle$ emitted by the agent. On the other hand, it can be verified that $P/\mathcal{M}_H$ leads to the same result. The intuition is that since each high-level operation is hidden, then the agent evolves, through a number of internal steps, into state $\langle l\rangle$. Therefore, $P \in$ *rd*-BSNNI. However, if the high-level user consumes the tuple $\langle h\rangle$ emitted by P, thus preventing the execution of the input operation $in(h)$, then the agent reaches a deadlock state without emitting the tuple $\langle l\rangle$. Such a behavior is captured by the *rd*-BNNI property. Indeed, agent $((\langle h\rangle \mid in(h).out(l))\backslash_I \mathcal{M}_H)/\mathcal{M}_H$, which is reachable from $(P\backslash_I \mathcal{M}_H)/\mathcal{M}_H$, enables a τ labelled transition obtained by hiding the action $\bar{h}$, representing the output of tuple $\langle h\rangle$. This transition leads to agent $((in(h).out(l))\backslash_I \mathcal{M}_H)/\mathcal{M}_H$, where the execution of the input operation is prevented. The effect of such a behavior is that $P \notin$ *rd*-BNNI. The problem revealed by the *rd*-BNNI property would be solved by requiring h to be a local name of agent P that cannot be consumed by any high-level user. For instance, we have that $P\backslash h \in$ *rd*-BNNI, *rd*-BSNNI. On the contrary, in the synchronous setting of [4], P is neither BNNI nor BSNNI.

Example 3.8 Let us consider the following slight modification of the agent of Example 3.7:

$$Q \stackrel{def}{=} out(h).in(h).out(l) + in(h).$$

On the one hand, we have $Q \in$ *rd*-BNNI. Indeed, both $(Q\backslash_I \mathcal{M}_H)/\mathcal{M}_H$ and $Q/\mathcal{M}_H$ are weakly *rd*-bisimulation equivalent to process $\tau + \tau.out(l)$. Intuitively, $Q/\mathcal{M}_H$ can either evolve into a deadlock state by performing the right-hand term of the alternative choice operator, or execute a process that, as seen in Example 3.7, leads to the emission of tuple $\langle l \rangle$. Moreover, agent $(Q\backslash_I \mathcal{M}_H)/\mathcal{M}_H$ behaves as agent $(P\backslash_I \mathcal{M}_H)/\mathcal{M}_H$ of Example 3.7, i.e. it can either deadlock without emitting the message l or output the tuple $\langle l \rangle$. On the other hand, it can be verified that $Q \notin$ *rd*-BSNNI. In particular, the semantics of $Q\backslash\mathcal{M}_H$ is the same as that of $P\backslash\mathcal{M}_H$ of Example 3.7, which cannot be equivalent to $\tau+\tau.out(l)$, as $P\backslash\mathcal{M}_H$ is forced to emit the tuple $\langle l \rangle$. Differently from Example 3.7, in this case *rd*-BNNI is not able to reveal a covert channel. Similarly as seen in Example 3.7, note that $Q\backslash h \in$ *rd*-BNNI, *rd*-BSNNI. Finally, we also point out that, in the synchronous setting of [4], Q is neither BNNI nor BSNNI.

As we have seen in the examples above, *rd*-BNNI and *rd*-BSNNI do not match whenever the agent can consume the tuples that it offers to the environment. Indeed, in this case the interleaving between the high-level user operations and the agent activities interferes with the low-level view of the tuplespace. We can avoid such a kind of interference if we assume that the agent distinguishes between tuples offered to the environment (which, therefore, cannot be consumed by the agent itself) and tuples that the agent employs for its internal calculations (which must be modeled through local names).

Definition 3.9 An agent $P \in \mathcal{A}$ is said to be a *non-consuming producer* (*ncp*, for short) if $\forall P'.\, P \Rightarrow P'$, $P' \stackrel{\bar{a}}{\rightarrow}$ implies that P' does not enable a τ labelled transition that derives from a synchronization involving label $\bar{a}$.

For instance, process P of Example 3.7 is not a *ncp* agent. Indeed, as we have seen, P can first emit a tuple $\langle h \rangle$, and then can either offer it to the environment or consume it via a corresponding input operation. On the other hand, process $P\backslash h$ is a *ncp* agent, since, differently from P, it cannot offer the tuple $\langle h \rangle$ to the environment just after emitting it. We can argue similarly for the agent Q described in Example 3.8.

Definition 3.9 allows *rd*-BNNI and *rd*-BSNNI to be related, as stated by the following proposition. Instead, we recall that in the synchronous communication setting BNNI $\not\subseteq$ BSNNI and BSNNI $\not\subseteq$ BNNI [4].

Proposition 3.10 *If $P \in \mathcal{A}$ is a non-consuming producer, then*

$$P \in \textit{rd}\text{-BNNI} \Leftrightarrow \mathrm{P} \in \textit{rd}\text{-BSNNI}.$$

Proof. Let P be an agent and R be a relation defined as follows:
$((P'\backslash_I \mathcal{M}_H)/\mathcal{M}_H, P'\backslash\mathcal{M}_H) \in R$, $\forall P'.\, P \Rightarrow P'$.
We now prove that R is a weak *rd*-bisimulation up to $\approx_{\mathrm{B}_{rd}}$[3]. By inspection

[3] The definition is a straightforward extension of that of [14].

of possible cases, it follows:

- if $P'\backslash\mathcal{M}_H \stackrel{\alpha}{\rightarrow} P''\backslash\mathcal{M}_H$ then $(P'\backslash_I\mathcal{M}_H)/\mathcal{M}_H \stackrel{\alpha}{\rightarrow} (P''\backslash_I\mathcal{M}_H)/\mathcal{M}_H$, since the transitions enabled in $P'\backslash\mathcal{M}_H$ are a subset of the transitions enabled in $(P'\backslash_I\mathcal{M}_H)/\mathcal{M}_H$;
- if $(P'\backslash_I\mathcal{M}_H)/\mathcal{M}_H \stackrel{\alpha}{\rightarrow} (P''\backslash_I\mathcal{M}_H)/\mathcal{M}_H$, where α is either a visible action or a τ action that is not obtained by hiding a $\bar{h}$ labelled transition enabled in $P'\backslash_I\mathcal{M}_H$, then $P'\backslash\mathcal{M}_H \stackrel{\alpha}{\rightarrow} P''\backslash\mathcal{M}_H$, since α models an event of term P' that is not restricted in $P'\backslash\mathcal{M}_H$.
- if $P' = \langle h\rangle|P''$ and $((\langle h\rangle|P'')\backslash_I\mathcal{M}_H)/\mathcal{M}_H \stackrel{\tau}{\rightarrow} (P''\backslash_I\mathcal{M}_H)/\mathcal{M}_H$, then, since P is a *ncp* agent, we have that $(\langle h\rangle|P'')\backslash\mathcal{M}_H \equiv (\underline{0}|P'')\backslash\mathcal{M}_H \equiv P''\backslash\mathcal{M}_H$.

Therefore, R is a weak *rd*-bisimulation up to $\approx_{\mathrm{B}_{rd}}$ and $P\backslash\mathcal{M}_H \approx_{\mathrm{B}_{rd}} (P\backslash_I\mathcal{M}_H)/\mathcal{M}_H$, from which we immediately derive the result. □

An expected result is given by the following lemma, which confirms that the high-level outputs do not interfere with the observable behavior of a *ncp* agent.

Lemma 3.11 *If $P \in \mathcal{A}$ is a non-consuming producer, then*

$$P/_O\,\mathcal{M}_H \approx_{\mathrm{B}_{rd}} P\backslash_O\,\mathcal{M}_H.$$

Proof. A straightforward application of the proof of Proposition 3.10. □

We point out that the restriction to non-consuming producers is reasonable in an asynchronous setting, since in practice such an assumption states that once a signal (output communication) is emitted by the agent, then the environment can eventually consume it.

In general, we can summarize the presentation of the noninterference properties by emphasizing the following results, as also graphically reported in Fig. 3.

- In a synchronous setting SNNI is stricter than NNI, while BSNNI and BNNI do not satisfy any inclusion relation. In an asynchronous setting *rd*-NNI and *rd*-SNNI turn out to have the same expressive power. Such an equivalence holds also when passing to the weak *rd*-bisimulation equivalence, provided that we restrict ourselves to the set of non-consuming producers.
- NNI and *rd*-NNI have the same expressive power in the sense that they capture the same illegal interferences, i.e. those that are caused by high-level inputs. This is because in both communication models input is a blocking operation.
- As a consequence of the considerations above, SNNI is stricter than *rd*-SNNI. This is because the asynchronous assumption prevents the high-level user from setting up a covert channel based on output operations.

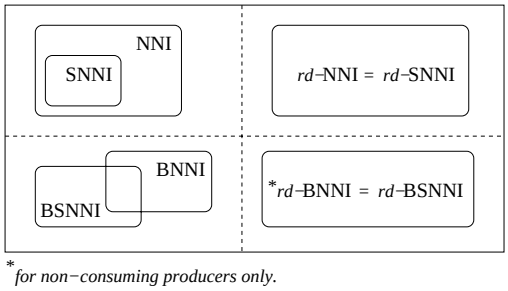

Fig. 3. Relation among properties in synchronous/asynchronous models.

3.3 The Access Monitor Example

In this section, we consider as a simple case study an access monitor (similarly as done in [6]) and we show (i) the kind of information flow that can be revealed in the asynchronous setting, and (ii) the main differences with respect to an approach based on a synchronous communication model.

Let us consider system *Sys* of Table 3 representing an access monitor that handles read and write requests on a low-level binary variable enforcing the multi-level security policy [2]. We recall that such a policy says that a process at σ-security level may only (i) write variables at the same level or above (*write up*), and (ii) read variables at the same level or below (*read down*). Agent *Monitor* accepts all the user requests and has direct access to agent *Low_Bit*, which in turn handles the binary variable, whose initial value is 0 (see tuple $\langle b_0 \rangle$). Moreover, the interactions between *Monitor* and *Low_Bit* and the operations internally performed by *Low_Bit* are local (i.e., any user cannot offer/consume the related tuples, as imposed by the restriction operators in agent *Sys*). Agent *Monitor* interacts with the environment by consuming the messages low_r, denoting a low-level read request, $high\_r$, denoting a high-level read request, low_w_0 and low_w_1, denoting low-level write requests, and by emitting the high- (low-) level messages h_i (l_i), for $i \in \{0, 1\}$, denoting that the value i is communicated to the environment. Note that, according to the multi-level security policy, high-level write requests ($high\_w_0, high\_w_1$) are not satisfied, since a high-class user cannot write a low-security variable.

In the following, we assume that a single low-level user interacts with the system and we show what such a user can learn about the high-security operations by manipulating the low-level variable. The noninterference analysis reveals that $Sys \in$ *rd*-BSNNI. Moreover, we point out that *Sys* is a *ncp* agent. Therefore, it satisfies also the *rd*-BNNI property. However, if we allow agent *Monitor* to satisfy high-level write requests, then the *write up* condition is violated. Formally, if we add to the algebraic specification the alternative behaviors

$$in(high\_w_0).out(w_0).in(ack).Monitor +$$
$$in(high\_w_1).out(w_1).in(ack).Monitor$$

$$
\begin{aligned}
&Low\_Bit \stackrel{def}{=} \\
&\quad in(r).(rd(b_0).out(r_0).Low\_Bit + rd(b_1).out(r_1).Low\_Bit) + \\
&\quad in(w_0).(rd(b_0).out(ack).Low\_Bit + in(b_1).out(b_0).out(ack).Low\_Bit) + \\
&\quad in(w_1).(rd(b_1).out(ack).Low\_Bit + in(b_0).out(b_1).out(ack).Low\_Bit) \\
&Monitor \stackrel{def}{=} \\
&\quad in(low\_r).out(r).(in(r_0).out(l_0).Monitor + in(r_1).out(l_1).Monitor) + \\
&\quad in(high\_r).out(r).(in(r_0).out(h_0).Monitor + in(r_1).out(h_1).Monitor) + \\
&\quad in(low\_w_0).out(w_0).in(ack).Monitor + \\
&\quad in(low\_w_1).out(w_1).in(ack).Monitor + \\
&\quad in(high\_w_0).Monitor + in(high\_w_1).Monitor \\
&Sys \stackrel{def}{=} (Monitor \,|\, (Low\_Bit \,|\, \langle b_0 \rangle) \backslash \{b_0, b_1\}) \backslash \{r, r_0, r_1, w_0, w_1, ack\}
\end{aligned}
$$

Table 3
Access monitor example

then neither *rd*-BSNNI nor *rd*-SNNI are satisfied. For both properties, the equivalence checking captures the fact that, e.g., the low-level user perceives the high-level interference by first reading l_0 and then reading l_1 in two consecutive read operations.

The same example, based on a synchronous communication model, is presented in [6], where it is shown that the corresponding properties hold under the same conditions.

Now, assume that a high-level output is added to inform the high-security user that a low-level write operation occurred. This can be modeled by adding the high-level output $out(written\_i)$, for $i \in \{0, 1\}$, to agent *Monitor* as follows:

$$
\begin{aligned}
&in(low\_w_0).out(w_0).in(ack).out(written\_0).Monitor + \\
&in(low\_w_1).out(w_1).in(ack).out(written\_1).Monitor.
\end{aligned}
$$

In the synchronous setting such a version of the monitor does not satisfy the SNNI property [6], since the high-security user can refuse the feedback concerning the low-level write operation. Hence, two consecutive low-level write operations can be exploited to inform the low-security user that the high-class user is still active. Such a behavior is more than enough to set up a 1-bit covert channel from high level to low level. In our asynchronous setting, it can be verified that such a version of the access monitor is still secure. This

is because the high-level user cannot block or delay the output offered by the monitor, which is interpreted as an independent activity (a signal) instead of a synchronous, direct interaction with the high-level user.

4 Related Work and Conclusion

In this paper, we presented two semantics for the formalization of noninterference properties in the context of a process algebra with asynchronous communication based on Linda-like coordination primitives. The security properties, borrowed from [4], are based on both a trace semantics and a bisimulation semantics. To the best of our knowledge, this work represents the first effort to combine the noninterference approach to information flow theory and the Linda coordination model. Along these lines of investigation, a work on asynchronous communication in process algebras and a variety of security properties has been recently done by Hennessy and Riely in [9]. In particular, they define an extension of the asynchronous π-calculus in which resource access control and secure information flow can be verified using types. The type system that is proposed for establishing noninterference works if the notion of process behavior, necessary to formalize a noninterference result, is restricted to a *may* testing equivalence.

As a future work, interesting results might be obtained by employing well-established equational theories developed for asynchronous calculi (see, e.g., [10,1,11]). Moreover, as far as the noninterference analysis is concerned, the classification of security properties of [4] includes many more definitions that express a slightly different intuition of what noninterference means. Among them we cite Nondeducibility on Composition (NDC), which states that an active high-level agent interacting with the system has not to alter the low-level behavior of the system. In such a framework, the active agent is formalized as a process that performs high-level actions only and is put in parallel with the system. Because of the synchronous nature of the communication model, the active agent decides which input/output operations of the system will be chosen for execution and which ones, instead, will never occur. However, in a Linda-like communication model, the only interesting interactions between the system and the active agent concern the high-level input operations that the system is allowed to perform. Such observations lead us to consider the Nondeducibility on Strategies property (NDS) of [15], whose main difference with respect to NDC is given by the synchronous nature of the output operation adopted in [4]. In particular, a strategy is a function that, looking at previous high-level inputs and outputs, decides the new high-level input. Then, the NDS basic idea is that the system, when composed to any strategy, has not to alter its behavior as observed by a low-level observer. On the basis of such a principle, we plan to redefine in our setting a NDC-like property that expresses the same intuition as that behind the notion of NDS.

References

[1] Amadio, R., I. Castellani, and D. Sangiorgi, *On Bisimulations for the Asynchronous π–Calculus*, Theoretical Computer Science **195**(2) (1998), 291–324.

[2] Bell, D. E., and L. J. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", Technical Report ESD-TR-75-306 (1976), The MITRE Corp., Bedford (MA).

[3] Busi, N., R. Gorrieri, and G. Zavattaro, *A Process Algebraic View of Linda Coordination Primitives*, Theoretical Computer Science **192**(2) (1998), 167–199.

[4] Focardi, R., and R. Gorrieri, *A Classification of Security Properties*, Journal of Computer Security **3**(1) (1995), 5–33.

[5] "Foundations of Security Analysis and Design - Tutorial Lectures", R. Focardi and R. Gorrieri (Eds.), Springer LNCS **2171** (2001), 396 pp.

[6] Focardi, R., and R. Gorrieri, *Classification of Security Properties (Part I: Information Flow)*, in [5], 331–396.

[7] Goguen, J. A., and J. Meseguer, *Security Policy and Security Models*, Proc. IEEE Symp. on Security and Privacy (1982), 11–20.

[8] Guttman, J., and M. Nadel, *What Needs Securing?*, Proc. 1st Computer Security Foundation Workshop (1988), 34–57.

[9] Hennessy, M., and J. Riely, *Information Flow vs. Resource Access in the Asynchronous pi-calculus*, ACM Transactions on Programming Languages and Systems **24**(5) (2002), ACM Press, 566–591.

[10] Honda, K., and M. Tokoro, *On Asynchronous Communication Semantics*, Proc. ECOOP'91: Workshop on Object-Based Concurrent Computing, Springer-Verlag vol. **612** (1992), 21–51.

[11] Merro, M., and D. Sangiorgi, *On Asynchrony in Name-passing Calculi*, Proc. 25th Int. Colloquium on Automata, Languages, and Programming (ICALP'98), Springer LNCS **1443** (1998), 856–867.

[12] Milner, R., "Communication and Concurrency", Prentice Hall, 1989.

[13] Roscoe, A. W., *CSP and Determinism in Security Modelling*, Proc. IEEE Symp. on Security and Privacy (1995), 114–127.

[14] Sangiorgi, D., and R. Milner, *The Problem of Weak Bisimulation up to*, Proc. 3rd Int. Conf. on Concurrency Theory (CONCUR'92), Springer LNCS **630** (1992), 32–46.

[15] Wittbold, J. T., and D. M. Johnson, *Information Flow in Nondeterministic Systems*, Proc. Symposium on Research in Security and Privacy, IEEE CS Press (1990), 144–161.