Interleaving vs True Concurrency: Some Instructive Security Examples

Roberto Gorrieri(✉)

Dipartimento di Informatica—Scienza e Ingegneria, Università di Bologna,
Mura A. Zamboni, 7, 40127 Bologna, Italy
roberto.gorrieri@unibo.it

Abstract. Information flow security properties were defined some years ago in terms of suitable equivalence checking problems. These definitions were provided by using sequential models of computations (e.g., labeled transition systems [17,26]), and interleaving behavioral equivalences (e.g., bisimulation equivalence [27]). More recently, the distributed model of Petri nets has been used to study non-interference in [1,5,6], but also in these papers an interleaving semantics was used. By exploiting a simple process algebra, called CFM [18] and equipped with a Petri net semantics, we provide some examples showing that team equivalence, a truly-concurrent behavioral equivalence proposed in [19,20], is much more suitable to define information flow security properties. The distributed non-interference property we propose, called DNI, is very easily checkable on CFM processes, as it is compositional, so that it does not suffer from the state-space explosion problem. Moreover, DNI is characterized syntactically on CFM by means of a type system.

1 Introduction

The title of this paper reminds [8], where a strong argument is presented in support of the use of truly concurrent semantics: some of them, differently from interleaving semantics, can be a congruence for *action refinement* (see, e.g., the tutorial [15]). The purpose of this note is similar: to present one further argument in support of true concurrency, by showing by means of examples that, by adopting an interleaving semantics to describe distributed systems, some real information flows cannot be detected.

Information flow security properties were defined some years ago (see, e.g., the surveys [10,32]) in terms of suitable equivalence checking problems. Usually, a distributed system is described by means of a term of some process algebra (typically CCS [17,27] or CSP [23]), whose operational semantics is often given in terms of labeled transition systems (LTSs, for short) [26] and whose observational semantics is usually defined by means of some *interleaving* behavioral equivalence such as trace semantics or bisimulation semantics (see, e.g., the textbook [17] for an overview of behavioral equivalences on LTSs). More recently, the distributed model of Petri nets [18,31] was used to study non-interference in [1,5,6], but

R. Janicki et al. (Eds.): PETRI NETS 2020, LNCS 12152, pp. 131–152, 2020.
https://doi.org/10.1007/978-3-030-51831-8_7

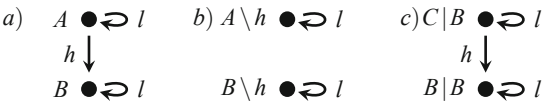

Fig. 1. An example of a secure process with never-ending behavior

also in these papers the security properties are based on interleaving behavioral semantics and the distributed model is used only to show that, under some conditions, these (interleaving) information flows can be characterized by the presence (or absence) of certain causal or conflict structures in the net.

The thesis of this paper is that, for security analysis, it is necessary to describe the behavior of distributed systems by means of a distributed model of computation, such as Petri nets, but also to observe the distributed model by means of some truly-concurrent behavioral semantics. There is a wide range of possible truly-concurrent equivalences (see, e.g., [13,14] for a partial overview) and it may be not obvious to understand which is more suitable. Our intuition is that, in order to capture all the possible information flows, it is necessary that the observational semantics is very concrete, observing not only the partial order of events that have occurred, as in *fully-concurrent bisimilarity* [4], but also the structure of the distributed state, as in *team equivalence* [19], *state-sensitive fully-concurrent bisimilarity* [22] and *structure-preserving bisimilarity* [14].

Our aim is to analyze systems that can perform two kinds of actions: *high-level* actions, representing the interaction of the system with high-level users, and *low-level* actions, representing the interaction with low-level users. We want to verify whether the interplay between the high user and the high part of the system can affect the view of the system as observed by a low user. We assume that the low user knows the structure of the system, and we check if, in spite of this, (s)he is not able to infer the behavior of the high user by observing the low view of the execution of the system. Hence, we assume that the set of actions is partitioned into two subsets: the set H of high-level actions and the set L of low-level actions.

To explain our point of view, we use the process algebra CFM [18,20], extending finite-state CCS [27] with a top-level operator of asynchronous (i.e., without communication capabilities) parallelism. Hence, the systems we are going to define, onto which high-level users and low-level users interact, are simply a collection of non-interacting, independent, sequential processes. Consider the CFM sequential process

$$A \doteq l.A + h.B \qquad B \doteq l.B$$

where A and B are constants, each equipped with a defining equation, and where l is a low action and h is a high one. The LTS semantics for A is in Fig. 1(a). Intuitively, this system is secure because the execution of the high-level action h does not add any information to a low-level observer: what such a low observer can see is just a sequence of the low action l in any case. The most restrictive non-

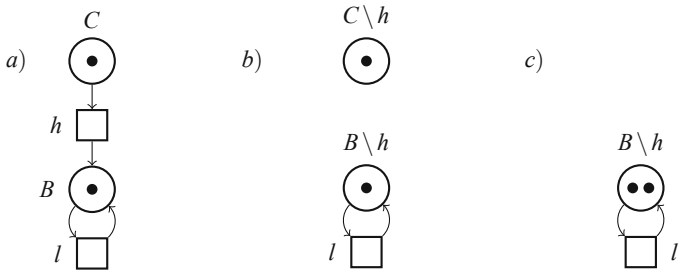

Fig. 2. The Petri net for the process $C \mid B$ in (a), of $(C \mid B)\backslash h$ in (b) and of $(B \mid B)\backslash h$ in (c)

interference property discussed in [6,10], called SBNDC, requires that whenever the system performs a high-level action, the states of the system *before* and *after* executing that high-level action are indistinguishable for a low-level observer. In our example, this means that A is SBNDC if $A\backslash h$ is bisimilar to $B\backslash h$, where $A\backslash h$ denotes the process A where the transition h is pruned. By observing the LTSs in Fig. 1(b), we conclude that A is SBNDC.

However, the LTS in Fig. 1(a) is isomorphic to the LTS in 1(c), which is the semantics of the parallel process $C \mid B$, where $C \doteq h.B$. Therefore, we can conclude that also $C \mid B$ enjoys the SBNDC non-interference property. Unfortunately, this parallel system is not secure: if a low observer can realize that two occurrences of actions l have been performed in parallel (which is a possible behavior of $B \mid B$, but this behavior is not represented in the LTS semantics),

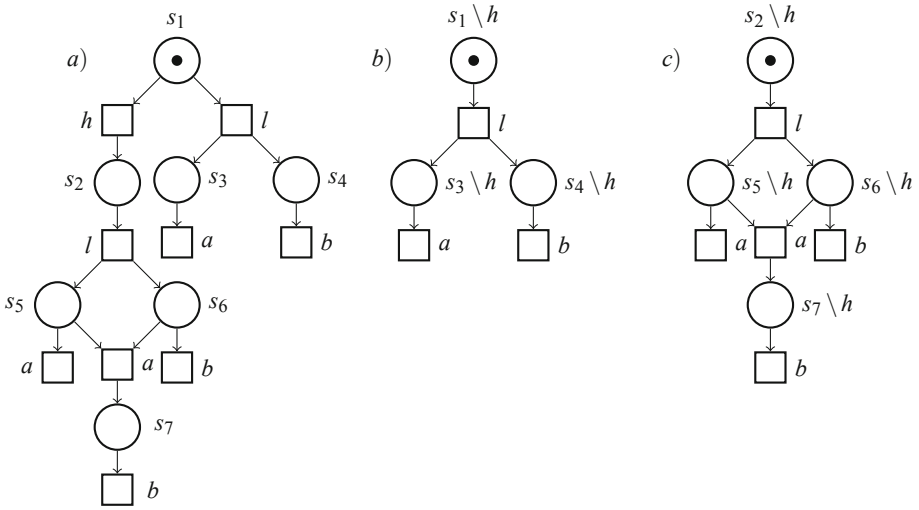

Fig. 3. An example of an insecure process

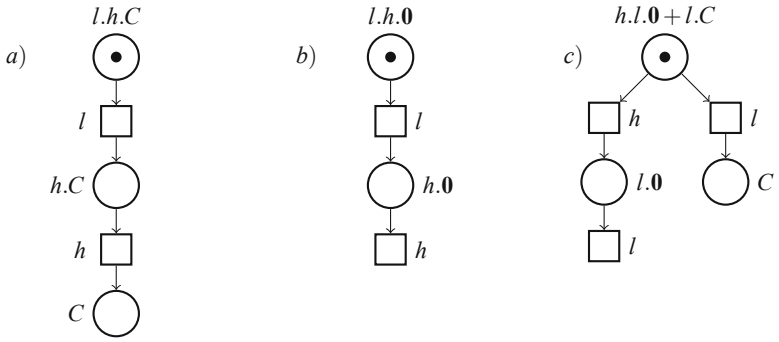

Fig. 4. By observing the state, information flows can be detectable

then (s)he is sure that the high-level action h has been performed. This trivial example show that SBNDC, based on interleaving bisimulation equivalence, is unable to detect a real information flow from high to low.

However, if we use a Petri net semantics for the process algebra CFM, as described in Sect. 3, we have that the net semantics for $C \mid B$, outlined in Fig. 2(a), is such that $(C \mid B)\backslash h$ (whose net semantics is outlined in Fig. 2(b)) is not "truly-concurrent" bisimilar to $(B \mid B)\backslash h$ (whose net semantics is in Fig. 2(c)). In fact, it is now visible that the two tokens on place $B\backslash h$ in the net in (c) can perform transition l at the same time, and such a behavior is impossible for the net in (b). This example calls for a truly-concurrent semantics which is at least as discriminating as *step bisimulation* equivalence [18,28], which is able to observe the parallel execution of actions.

However, it is not difficult to elaborate on the example above in order to show that, actually, a more discriminating truly-concurrent semantics is necessary. E.g., the net in Fig. 3(a) is such that the two low-observable subnets *before* and *after* the execution of h (depicted in Fig. 3(b) and 3(c), resp.) are step bisimilar, even if they generate different partial orders: if an observer realizes that b is causally dependent on a, then (s)he infers that h has been performed. (To be precise, this example is not expressible in our simple CFM process algebra, but it can in the more generous process algebra FNM [18].) One of the most accurate (and popular) truly-concurrent behavioral equivalences for Petri nets is *fully-concurrent bisimilarity* [4] (fc-bisimilarity, for short), whose intuition originates from *history-preserving bisimilarity* [13]. So, it seems that we can simply change the SBNDC definition by replacing interleaving bisimilarity (on the LTS semantics) with fc-bisimilarity (on the Petri net semantics). However, fc-bisimilarity observes only the partial order of the performed events, while it abstracts from the size of the distributed state, because it may relate markings composed of a completely different number of tokens. The following example shows that also the size of the marking is an important feature, that cannot be ignored, as fc-bisimilarity does.

Consider the net in Fig. 4(a), which is the semantics of the CFM process $l.h.C$, with $C \doteq \mathbf{0}$. Since h is not causing any low action, we conclude that this net is secure. However, the very similar net in (b), which is the semantics of $l.h.\mathbf{0}$, is not secure: if the low observer realizes that the token disappears in the end, then (s)he can infer that h has been performed. This simple observation is about the observability of the size of the distributed system: if the execution of a high-level action modifies the current number of tokens (i.e., the number of currently active sequential subprocesses), then its execution has an observable effect that can be recognized by a low observer. Similarly, the net in (c), which is the semantics of $h.l.\mathbf{0} + l.C$ with $C \doteq \mathbf{0}$, is such that, before and after h, the same partial order of event can be performed, but if the token disappears in the end, then the low observer knows that h has been performed. Therefore, it is necessary to use a truly-concurrent behavioral equivalence slightly finer than fully-concurrent bisimilarity, able to observe also the structure of the distributed state, such as *state-sensitive fc-bisimilarity* [22] and *structure-preserving bisimilarity* [14], which can be characterized in a very simple and effective way on CFM as *team equivalence* [19,20,22]. So, we are now ready to propose the property DNI (acronym of Distributed Non-Interference) as follows: a CFM process p is DNI if for each reachable p', p'' and for each $h \in H$ such that $p' \xrightarrow{h} p''$, we have $p'\backslash H \sim^{\oplus} p''\backslash H$, where $\sim^{\oplus}$ denotes team equivalence.

The DNI check can be done in a simple way. Given a process $p = p_1 \,|\, p_2 \,|\, \ldots \,|\, p_n$, where each p_i is sequential, then we prove that p is DNI if and only if each p_i is DNI. Hence, instead of inspecting the state-space of p, we can simply inspect the state-spaces for $p_1, p_2, \ldots p_n$. If the state-space of each p_i is composed of 10 states, then the state-space of p is composed of 10^n states, so that a direct check of DNI on the state space of p is impossible for large values of n. However, the distributed analysis on each p_i can be done in linear time w.r.t. n, hence also for very large values of n. Moreover, a structural characterization of DNI can be provided, as well as a typing system: we prove that a CFM process p is DNI iff p (or a slight variant of it) is typed. The typing system is based on a finite, sound and complete, axiomatization of team equivalence [20].

The paper is organized as follows. Section 2 introduces the simple Petri net model we are using, namely *finite-state machines*, together with the definitions of *bisimulation on places* and *team equivalence*. Section 3 defines the syntax of the process algebra CFM, its net semantics and recalls the finite, sound and complete, axiomatization of team equivalence from [20]. Section 4 introduces the distributed non-interference property DNI on CFM processes, proving that it can be really checked in a distributed manner, and also describes the typing system for DNI. Finally, Sect. 5 comments on related work, outlines some possible future research and lists some additional reasons to prefer truly-concurrent semantics over interleaving semantics.

2 Finite-State Machines and Team Equivalence

By finite-state machine (FSM, for short) we mean a simple type of finite Petri net [18,31] whose transitions have singleton pre-set and singleton, or empty, post-set.

The name originates from the fact that an unmarked net of this kind is essentially isomorphic to a nondeterministic finite automaton [24] (NFA, for short), usually called a finite-state machine as well. However, semantically, our FSMs are richer than NFAs because, as their initial marking may be not a singleton, these nets can also exhibit concurrent behavior, while NFAs are strictly sequential.

Definition 1 (Multiset). *Let $\mathbb{N}$ be the set of natural numbers. Given a finite set S, a* multiset *over S is a function $m : S \to \mathbb{N}$. Its* support *set $dom(m)$ is $\{s \in S \mid m(s) \neq 0\}$. The set of all multisets over S is $\mathscr{M}(S)$, ranged over by m. We write $s \in m$ if $m(s) > 0$. The* multiplicity *of s in m is the number $m(s)$. The* size *of m, denoted by $|m|$, is the number $\sum_{s \in S} m(s)$, i.e., the total number of its elements. A multiset m such that $dom(m) = \emptyset$ is called* empty *and is denoted by θ. We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$.*

Multiset union $\_ \oplus \_$ *is defined as follows: $(m \oplus m')(s) = m(s) + m'(s)$; it is commutative, associative and has θ as neutral element.* Multiset difference $\_ \ominus \_$ *is defined as follows: $(m_1 \ominus m_2)(s) = max\{m_1(s) - m_2(s), 0\}$. The* scalar product *of a number j with m is the multiset $j \cdot m$ defined as $(j \cdot m)(s) = j \cdot (m(s))$. By s_i we also denote the multiset with s_i as its only element. Hence, a multiset m over $S = \{s_1, \ldots, s_n\}$ can be represented as $k_1 \cdot s_1 \oplus k_2 \cdot s_2 \oplus \ldots \oplus k_n \cdot s_n$, where $k_j = m(s_j) \geq 0$ for $j = 1, \ldots, n$.* □

Definition 2 (Finite-state machine). *A labeled* finite-state machine *(FSM, for short) is a tuple $N = (S, A, T)$, where*

- *S is the finite set of* places*, ranged over by s (possibly indexed),*
- *A is the finite set of* labels*, ranged over by ℓ (possibly indexed), and*
- *$T \subseteq S \times A \times (S \cup \{\theta\})$ is the finite set of* transitions*, ranged over by t.*

Given a transition $t = (s, \ell, m)$, we use the notation ${}^\bullet t$ to denote its pre-set *s (which is a single place) of tokens to be consumed; $l(t)$ for its* label *ℓ, and $t^\bullet$ to denote its* post-set *m (which is a place or the empty multiset θ) of tokens to be produced. Hence, transition t can be also represented as ${}^\bullet t \xrightarrow{l(t)} t^\bullet$.* □

Definition 3 (Marking, FSM net system). *A multiset over S is called a* marking*. Given a marking m and a place s, we say that the place s contains $m(s)$* tokens*, graphically represented by $m(s)$ bullets inside place s. An* FSM net system *$N(m_0)$ is a tuple (S, A, T, m_0), where (S, A, T) is an FSM and m_0 is a marking over S, called the* initial marking*. We also say that $N(m_0)$ is a* marked *net. An FSM net system $N(m_0) = (S, A, T, m_0)$ is* sequential *if m_0 is a singleton, i.e., $|m_0| = 1$; while it is* concurrent *if m_0 is arbitrary.* □

Definition 4 (Firing sequence, reachable markings). *Given an FSM $N = (S, A, T)$, a transition t is* enabled *at marking m, denoted by $m[t\rangle$, if ${}^\bullet t \subseteq m$. The execution (or* firing*) of t enabled at m produces the marking $m' = (m \ominus {}^\bullet t) \oplus t^\bullet$. This is written usually as $m[t\rangle m'$, but also as $m \xrightarrow{l(t)} m'$. A* firing sequence *starting at m is defined inductively as*

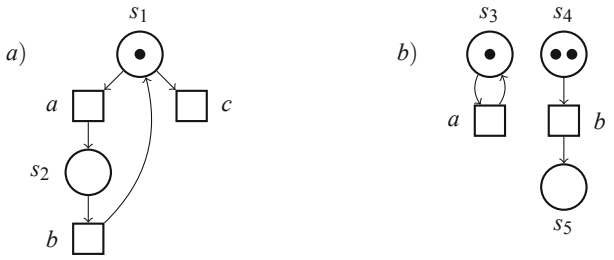

Fig. 5. A sequential finite-state machine in (a), and a concurrent finite-state machine in (b)

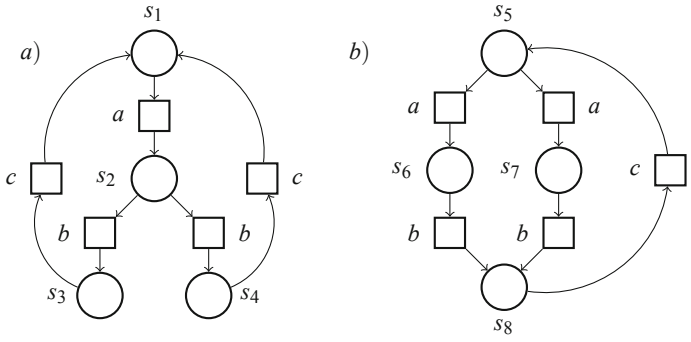

Fig. 6. Two bisimilar FSMs

- $m[\varepsilon\rangle m$ *is a firing sequence (where* ε *denotes an empty sequence of transitions) and*
- *if* $m[\sigma\rangle m'$ *is a firing sequence and* $m'[t\rangle m''$*, then* $m[\sigma t\rangle m''$ *is a firing sequence.*

If $\sigma = t_1 \dots t_n$ *(for* $n \geq 0$*) and* $m[\sigma\rangle m'$ *is a firing sequence, then there exist* $m_1, \dots, m_{n+1}$ *such that* $m = m_1[t_1\rangle m_2[t_2\rangle \dots m_n[t_n\rangle m_{n+1} = m'$*, and* $\sigma = t_1 \dots t_n$ *is called a* transition sequence *starting at* m *and ending at* m'*. The set of* reachable markings *from* m *is* $reach(m) = \{m' \mid \exists \sigma . m[\sigma\rangle m'\}$. □

Definition 5. *An FSM system* $N(m_0) = (S, A, T, m_0)$ *is* dynamically reduced *if* $\forall s \in S \ \exists m \in reach(m_0).m(s) \geq 1$ *and* $\forall t \in T \, \exists m, m' \in reach(m_0)$ *such that* $m[t\rangle m'$. □

Example 1. By using the usual drawing convention for Petri nets, Fig. 5 shows in (a) a sequential FSM, which performs a, possibly empty, sequence of a's and b's, until it performs one c and then stops *successfully* (the token disappears in the end). Note that a sequential FSM is such that any reachable marking is a singleton or empty. Hence, a sequential FSM is a *safe* (or 1-bounded) net: each place in any reachable marking can hold one token at most. In (b), a concurrent

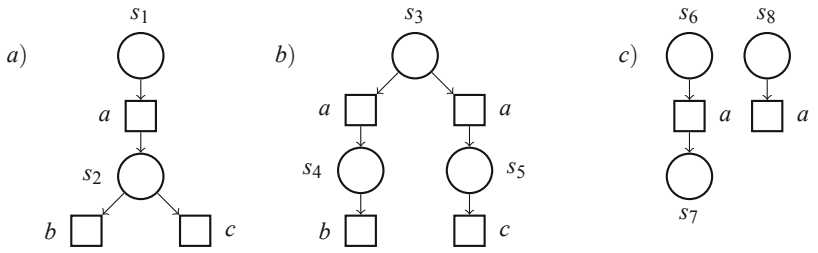

Fig. 7. Some (non-)-bisimilar FSMs

FSM is depicted: it can perform a forever, interleaved with two occurrences of b, only: the two tokens in s_4 will eventually reach s_5, which is a place representing unsuccessful termination (deadlock). Note that a concurrent FSM is a *k-bounded* net, where k is the size of the initial marking: each place in any reachable marking can hold k tokens at most. Hence, the set $reach(m)$ is finite for any m. As a final comment, we mention that for each FSM $N = (S, A, T)$ and each place $s \in S$, the set $reach(s)$ is a subset of $S \cup \{\theta\}$. □

2.1 Bisimulation on Places

We provide the definition of (strong) *bisimulation on places* for unmarked FSMs, originally introduced in [19]. Because of the shape of FSMs transitions, in the definition below the post-sets m_1 and m_2 can be either the empty marking θ or a single place.

Definition 6 (Bisimulation on places). *Let $N = (S, A, T)$ be an FSM. A* bisimulation *is a relation $R \subseteq S \times S$ such that if $(s_1, s_2) \in R$ then for all $\ell \in A$*

- *$\forall m_1$ such that $s_1 \xrightarrow{\ell} m_1$, $\exists m_2$ such that $s_2 \xrightarrow{\ell} m_2$ and either $m_1 = \theta = m_2$ or $(m_1, m_2) \in R$,*
- *$\forall m_2$ such that $s_2 \xrightarrow{\ell} m_2$, $\exists m_1$ such that $s_1 \xrightarrow{\ell} m_1$ and either $m_1 = \theta = m_2$ or $(m_1, m_2) \in R$.*

Two places s and s' are bisimilar *(or* bisimulation equivalent*), denoted $s \sim s'$, if there exists a bisimulation R such that $(s, s') \in R$.* □

Example 2. Consider the nets in Fig. 6. It is not difficult to realize that relation $R = \{(s_1, s_5), (s_2, s_6), (s_2, s_7), (s_3, s_8), (s_4, s_8)\}$ is a bisimulation on places. □

Example 3. Consider the nets in Fig. 7. It is not difficult to realize that $s_1 \not\sim s_3$. In fact, s_1 may reach s_2 by performing a; s_3 can reply to this move in two different ways by reaching either s_4 or s_5; however, while s_2 offers both b and c, s_4 may perform only b and s_5 only c; hence, s_4 and s_5 are not bisimilar to s_2 and so also s_1 is not bisimilar to s_3. This example shows that bisimulation

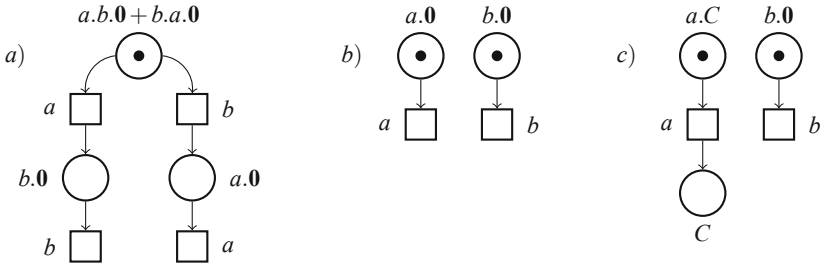

Fig. 8. Three net systems: $a.b.\mathbf{0} + b.a.\mathbf{0}$, $a.\mathbf{0} \,|\, b.\mathbf{0}$ and $a.C \,|\, b.\mathbf{0}$ (with $C \doteq \mathbf{0}$)

equivalence is sensitive to the timing of choice. Furthermore, also s_6 and s_8 are not bisimilar. In fact, s_6 can reach s_7 by performing a, while s_8 can reply by reaching the empty marking, but $\theta \not\sim s_7$. This example shows that bisimulation equivalence is sensitive to the kind of termination of a process: even if s_7 is stuck, it is not equivalent to θ because the latter is the marking of a properly terminated process, while s_7 denotes a deadlock situation. □

Remark 1 **(Complexity).** If m is the number of net transitions and n of places, checking whether two places of an FSM are bisimilar can be done in $O(m \log (n+1))$ time, by adapting the algorithm in [30] for ordinary bisimulation on LTSs. Indeed, this very same algorithm can compute $\sim$ by starting from an initial partition of $S \cup \{\theta\}$ composed of two subsets: S and $\{\theta\}$. Bisimulation on places enjoys the same properties of bisimulation on LTSs, i.e., it is coinductive and with a fixed-point characterization. The largest bisimulation is an equivalence and can be used to minimize the net [19]. □

2.2 Team Equivalence

Definition 7 (Additive closure). *Given an FSM net $N = (S, A, T)$ and a* place relation $R \subseteq S \times S$*, we define a* marking relation $R^{\oplus} \subseteq \mathscr{M}(S) \times \mathscr{M}(S)$*, called the* additive closure *of R, as the least relation induced by the following axiom and rule.*

$$\frac{}{(\theta, \theta) \in R^{\oplus}} \qquad \frac{(m_1, m_2) \in R \quad (m_1', m_2') \in R^{\oplus}}{(m_1 \oplus m_1', m_2 \oplus m_2') \in R^{\oplus}}$$

□

Note that, by definition, two markings are related by $R^{\oplus}$ only if they have the same size; in fact, the axiom states that the empty marking is related to itself, while the rule, assuming by induction that m_1 and m_2 have the same size, ensures that $s_1 \oplus m_1$ and $s_2 \oplus m_2$ have the same size. An alternative way to define that two markings m_1 and m_2 are related by $R^{\oplus}$ is to state that m_1 can be represented as $s_1 \oplus s_2 \oplus \ldots \oplus s_k$, m_2 can be represented as $s_1' \oplus s_2' \oplus \ldots \oplus s_k'$

and $(s_i, s_i') \in R$ for $i = 1, \ldots, k$. Note also that if R is an equivalence relation, then $R^{\oplus}$ is also an equivalence relation.

Definition 8 (Team equivalence). *Given a finite-state machine* $N = (S, A, T)$ *and the bisimulation equivalence* $\sim \subseteq S \times S$*, we define* team equivalence $\sim^{\oplus} \subseteq \mathscr{M}(S) \times \mathscr{M}(S)$*, as the additive closure of* $\sim$*.* □

Example 4. Continuing Example 2 about Fig. 6, we have, e.g., that the marking $s_1 \oplus 2 \cdot s_2 \oplus s_3 \oplus s_4$ is team equivalent to any marking with one token in s_5, two tokens distributed over s_6 and s_7, and two tokens in s_8; e.g., $s_5 \oplus s_6 \oplus s_7 \oplus 2 \cdot s_8$ or $s_5 \oplus 2 \cdot s_7 \oplus 2 \cdot s_8$. Note that $m_1 = s_1 \oplus 2 \cdot s_2$ has the same size of $m_2 = 2 \cdot s_5 \oplus s_6$, but the two are not team equivalent; in fact, we can first match s_1 with one instance of s_5; then, one instance of s_2 with s_6; but, now, we are unable to find a match for the second instance of s_2, because the only element left in m_2 is s_5, and s_2 is not bisimilar to s_5. □

Example 5. Team equivalence is a truly concurrent equivalence. The FSM in Fig. 8(a) denotes the net for the sequential CFM process term $a.b.\mathbf{0} + b.a.\mathbf{0}$, which can perform the two actions a and b in either order. On the contrary, the concurrent FSM in (b) denotes the net for the parallel CFM process term $a.\mathbf{0}|b.\mathbf{0}$. Note that place $a.b.\mathbf{0} + b.a.\mathbf{0}$ is not team equivalent to the marking $a.\mathbf{0} \oplus b.\mathbf{0}$, because the two markings have different size. Nonetheless, $a.b.\mathbf{0} + b.a.\mathbf{0}$ and $a.\mathbf{0} \oplus b.\mathbf{0}$ are interleaving bisimilar. □

Example 6. If two markings m_1 and m_2 are interleaving bisimilar and have the same size, then they may be not team equivalent. For instance, consider Fig. 8(c), which denotes the net for the CFM process term $a.C \,|\, b.\mathbf{0}$, where C is a constant with empty body, i.e., $C \doteq \mathbf{0}$. Markings $a.\mathbf{0} \oplus b.\mathbf{0}$ and $a.C \oplus b.\mathbf{0}$ have the same size, they are interleaving bisimilar (actually, they are even fully concurrent bisimilar [4]), but they are not team equivalent. In fact, if $a.C \oplus b.\mathbf{0} \xrightarrow{a} C \oplus b.\mathbf{0}$, then $a.\mathbf{0} \oplus b.\mathbf{0}$ may try to respond with $a.\mathbf{0} \oplus b.\mathbf{0} \xrightarrow{a} b.\mathbf{0}$, but $C \oplus b.\mathbf{0}$ and $b.\mathbf{0}$ are not team bisimilar markings, as they have different size. □

Remark 2 **(Complexity).** Once the place relation $\sim$ has been computed once and for all for the given net in $O(m \cdot \log(n+1))$ time, the algorithm in [19] checks whether two markings m_1 and m_2 are team equivalent in $O(k^2)$ time, where k is the size of the markings. In fact, if $\sim$ is implemented as an adjacency matrix, then the complexity of checking if two markings m_1 and m_2 (represented as a list of places with multiplicities) are related by $\sim^{\oplus}$ is $O(k^2)$, because the problem is essentially that of finding for each element s_1 of m_1 a matching, $\sim$-related element s_2 of m_2. □

3 CFM: Syntax, Semantics, Axiomatization

3.1 Syntax

Let *Act* be a finite set of actions, ranged over by μ, partitioned into two subsets L and H of low-level actions and high-level ones, respectively. Let $\mathscr{C}$ be a finite

Table 1. Structural operational LTS semantics for CFM

(Pref)	$\dfrac{}{\mu.p \xrightarrow{\mu} p}$	(Cons)	$\dfrac{p \xrightarrow{\mu} p'}{C \xrightarrow{\mu} p'}\ C \doteq p$
(Sum_1)	$\dfrac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'}$	(Sum_2)	$\dfrac{q \xrightarrow{\mu} q'}{p+q \xrightarrow{\mu} q'}$
(Par_1)	$\dfrac{p \xrightarrow{\mu} p'}{p \,|\, q \xrightarrow{\mu} p' \,|\, q}$	(Par_2)	$\dfrac{q \xrightarrow{\mu} q'}{p \,|\, q \xrightarrow{\mu} p \,|\, q'}$

set of constants, disjoint from Act, ranged over by $A, B, C, \ldots$. The CFM *terms* (where CFM is the acronym of *Concurrent Finite-state Machines*) are generated from actions and constants by the following abstract syntax (with three syntactic categories):

$$\begin{array}{lll} s ::= \mathbf{0} \mid \mu.q \mid s+s & & \textit{guarded processes} \\ q ::= s \mid C & & \textit{sequential processes} \\ p ::= q \mid p \,|\, p & & \textit{parallel processes} \end{array}$$

where $\mathbf{0}$ is the empty process, $\mu.q$ is a process where action μ prefixes the residual q ($\mu.-$ is the *action prefixing* operator), $s_1 + s_2$ denotes the alternative composition of s_1 and s_2 ($-+-$ is the *choice* operator), $p_1 \,|\, p_2$ denotes the asynchronous parallel composition of p_1 and p_2 and C is a constant. A constant C may be equipped with a definition, but this must be a guarded process, i.e., $C \doteq s$. A term p is a CFM *process* if each constant in $Const(p)$ (the set of constants used by p; see [18] for details) is equipped with a defining equation (in category s). The set of CFM processes is denoted by $\mathscr{P}_{CFM}$, the set of its sequential processes, i.e., those in syntactic category q, by $\mathscr{P}^{seq}_{CFM}$ and the set of its guarded processes, i.e., those in syntactic category s, by $\mathscr{P}^{grd}_{CFM}$. By $sort(p) \subseteq Act$ we denote the set of all the actions occurring in p.

3.2 Semantics

The interleaving LTS semantics for CFM is given by the structural operational rules in Table 1. Note that each state of the LTS is a CFM process. As an example, the LTS for $C \,|\, B$, with $C \doteq h.B$ and $B \doteq l.B$, is described in Fig. 1(c). It is possible to prove that, for any p, the set of states reachable from p is finite [18].

The Petri net semantics for CFM, originally in [18], is such that the set S_{CFM} of pla-ces is the set of the sequential CFM processes, without $\mathbf{0}$, i.e., $S_{CFM} = \mathscr{P}^{seq}_{CFM} \backslash \{\mathbf{0}\}$. The decomposition function $dec : \mathscr{P}_{CFM} \to \mathscr{M}(S_{CFM})$, mapping process terms to markings, is defined in Table 2. An easy induction proves that for any $p \in \mathscr{P}_{CFM}$, $dec(p)$ is a finite multiset of sequential processes. Note that, if $C \doteq \mathbf{0}$, then $\theta = dec(\mathbf{0}) \neq dec(C) = \{C\}$. Note also that $\theta = dec(\mathbf{0}) \neq dec(\mathbf{0}+\mathbf{0}) = \{\mathbf{0}+\mathbf{0}\}$, which is a deadlock place.

Table 2. Decomposition function

$dec(\mathbf{0}) = \theta$	$dec(\mu.p) = \{\mu.p\}$
$dec(p + p') = \{p + p'\}$	$dec(C) = \{C\}$
$dec(p \mid p') = dec(p) \oplus dec(p')$	

Table 3. Denotational net semantics

$[\![\mathbf{0}]\!]_I = (\emptyset, \emptyset, \emptyset, \theta)$	
$[\![\mu.p]\!]_I = (S, A, T, \{\mu.p\})$	given $[\![p]\!]_I = (S', A', T', dec(p))$ and where
	$S = \{\mu.p\} \cup S',\ A = \{\mu\} \cup A',\ T = \{(\{\mu.p\}, \mu, dec(p))\} \cup T'$
$[\![p_1 + p_2]\!]_I = (S, A, T, \{p_1 + p_2\})$	given $[\![p_i]\!]_I = (S_i, A_i, T_i, dec(p_i))$ for $i = 1, 2$, and where
	$S = \{p_1 + p_2\} \cup S'_1 \cup S'_2$, with, for $i = 1, 2$,
	$S'_i = \begin{cases} S_i & \exists t \in T_i \text{ such that } t^\bullet(p_i) > 0 \\ S_i \setminus \{p_i\} & \text{otherwise} \end{cases}$
	$A = A_1 \cup A_2,\ T = T' \cup T'_1 \cup T'_2$, with, for $i = 1, 2$,
	$T'_i = \begin{cases} T_i & \exists t \in T_i \,.\, t^\bullet(p_i) > 0 \\ T_i \setminus \{t \in T_i \mid {}^\bullet t(p_i) > 0\} & \text{otherwise} \end{cases}$
	$T' = \{(\{p_1 + p_2\}, \mu, m) \mid (\{p_i\}, \mu, m) \in T_i, i = 1, 2\}$
$[\![C]\!]_I = (\{C\}, \emptyset, \emptyset, \{C\})$	if $C \in I$
$[\![C]\!]_I = (S, A, T, \{C\})$	if $C \notin I$, given $C \doteq p$ and $[\![p]\!]_{I \cup \{C\}} = (S', A', T', dec(p))$
	$A = A', S = \{C\} \cup S''$, where
	$S'' = \begin{cases} S' & \exists t \in T' \,.\, t^\bullet(p) > 0 \\ S' \setminus \{p\} & \text{otherwise} \end{cases}$
	$T = \{(\{C\}, \mu, m) \mid (\{p\}, \mu, m) \in T'\} \cup T''$ where
	$T'' = \begin{cases} T' & \exists t \in T' \,.\, t^\bullet(p) > 0 \\ T' \setminus \{t \in T' \mid {}^\bullet t(p) > 0\} & \text{otherwise} \end{cases}$
$[\![p_1 \mid p_2]\!]_I = (S, A, T, m_0)$	given $[\![p_i]\!]_I = (S_i, A_i, T_i, m_i)$ for $i = 1, 2$, and where
	$S = S_1 \cup S_2,\ A = A_1 \cup A_2,\ T = T_1 \cup T_2,\ m_0 = m_1 \oplus m_2$

Now we provide a construction of the net system $[\![p]\!]_\emptyset$ associated with process p, which is compositional and denotational in style. The details of the construction are outlined in Table 3. The mapping is parametrized by a set of constants that have already been found while scanning p; such a set is initially empty and it is used to avoid looping on recursive constants. The definition is syntax driven and also the places of the constructed net are syntactic objects, i.e., CFM sequential process terms. For instance, the net system $[\![a.\mathbf{0}]\!]_\emptyset$ is a net composed of one single marked place, namely process $a.\mathbf{0}$, and one single transition $(\{a.\mathbf{0}\}, a, \theta)$. A bit of care is needed in the rule for choice: in order to include only strictly necessary places and transitions, the initial place p_1 (or p_2) of the subnet $[\![p_1]\!]_I$ (or $[\![p_2]\!]_I$) is to be kept in the net for $p_1 + p_2$ only if there exists a transition reaching place p_1 (or p_2) in $[\![p_1]\!]_I$ (or $[\![p_2]\!]_I$), otherwise p_1 (or p_2) can be safely removed in the new net. Similarly, for the rule for constants.

We now list some properties of the semantics, whose proofs are in [18], which state that CFM really represents the class of FSMs.

Theorem 1 (Only Concurrent FSMs). *For each CFM process p, $[\![p]\!]_\emptyset$ is a concurrent finite-state machine.* □

Definition 9 (Translating Concurrent FSMs into CFM Process Terms). *Let $N(m_0) = (S, A, T, m_0)$—with $S = \{s_1, \ldots, s_n\}$, $A \subseteq Act$, $T = \{t_1, \ldots, t_k\}$, and $l(t_j) = \mu_j$—be a concurrent finite-state machine. Function $\mathscr{T}_{CFM}(-)$, from concurrent finite-state machines to CFM processes, is defined as*

$$\mathscr{T}_{CFM}(N(m_0)) = \underbrace{C_1 | \cdots | C_1}_{m_0(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m_0(s_n)}$$

where each C_i is equipped with a defining equation $C_i \doteq c_i^1 + \cdots + c_i^k$ (with $C_i \doteq \mathbf{0}$ if $k = 0$), and each summand c_i^j, for $j = 1, \ldots, k$, is equal to

- $\mathbf{0}$, *if* $s_i \notin {}^\bullet t_j$;
- $\mu_j.\mathbf{0}$, *if* ${}^\bullet t_j = \{s_i\}$ *and* $t_j^\bullet = \emptyset$;
- $\mu_j.C_h$, *if* ${}^\bullet t_j = \{s_i\}$ *and* $t_j^\bullet = \{s_h\}$. □

Theorem 2 (All Concurrent FSMs). *Let $N(m_0) = (S, A, T, m_0)$ be a dynamically reduced, concurrent finite-state machine such that $A \subseteq Act$, and let $p = \mathscr{T}_{CFM}(N(m_0))$. Then, $[\![p]\!]_\emptyset$ is isomorphic to $N(m_0)$.* □

Therefore, thanks to these results (proved in [18]), we can conclude that the CFM process algebra truly represents the class of FSMs. Hence, we can transfer the definition of team equivalence from FSMs to CFM process terms in a simple way.

Definition 10. *Two CFM processes p and q are team equivalent, denoted $p \sim^\oplus q$, if, by taking the (union of the) nets $[\![p]\!]_\emptyset$ and $[\![q]\!]_\emptyset$, we have that $dec(p) \sim^\oplus dec(q)$.* □

Of course, for sequential CFM processes, team equivalence $\sim^\oplus$ coincides with bisimilarity on places $\sim$.

Finally, as we are going to use an auxiliary restriction operator over CFM terms of the form $p\backslash H$, we define its net semantics as follows.

Definition 11 (Semantics of the auxiliary restriction operator). *Given a CFM process p, whose net semantics is $[\![p]\!]_\emptyset = (S, A, T, dec(p))$, we define the net associated to $p\backslash H$ as the net $[\![p\backslash H]\!]_\emptyset = (S', A', T', m)$ where*

- $S' = \{s\backslash H \mid s \in S\}$, *i.e., each place is decorated by the restriction operator;*
- $A' = A\backslash H$, *i.e.,* $\{\mu \mid \mu \in A, \mu \notin H\}$;
- $T' = \{({}^\bullet t\backslash H, l(t), t^\bullet\backslash H) \mid t \in T, l(t) \notin H\}$;
- $m = dec(p)\backslash H$, *where the restriction operator is applied element-wise to the places, if any, of the marking $dec(p)$.* □

As an example, the net for $C \mid B$, with $C \doteq h.B$ and $B \doteq l.B$, is outlined in Fig. 2(a), while, assuming that H is composed of a single action h, the net for $(C \mid B)\backslash h$ is in Fig. 2(b).

3.3 Axiomatization

In this section we recall the sound and complete, finite axiomatization of team equivalence over CFM outlined in [20]. For simplicity's sake, the syntactic definition of *open* CFM (i.e., CFM with variables) is given with only one syntactic category, but each ground instantiation of an axiom must respect the syntactic definition of CFM given (by means of three syntactic categories) in Sect. 3.1; this means that we can write the axiom $x + (y + z) = (x + y) + z$, but it is invalid to instantiate it to $C + (a.\mathbf{0} + b.\mathbf{0}) = (C + a.\mathbf{0}) + b.\mathbf{0}$ because these are not legal CFM processes (the constant C cannot be used as a summand).

The set of axioms are outlined in Table 4. We call E the set of axioms {**A1, A2, A3, A4, R1, R2, R3, P1, P2, P3**}. By the notation $E \vdash p = q$ we mean that there exists an equational deduction proof of the equality $p = q$, by using the axioms in E. Besides the usual equational deduction rules of reflexivity, symmetry, transitivity, substitutivity and instantiation (see, e.g., [17]), in order to deal with constants we need also the following recursion congruence rule:

$$\frac{p = q \ \wedge \ A \doteq p\{A/x\} \ \wedge \ B \doteq q\{B/x\}}{A = B}$$

where $p\{A/x\}$ denotes the open term p where all occurrences of the variable x are replaced by A. The axioms **A1**–**A4** are the usual axioms for choice where, however, **A3**–**A4** have the side condition $x \neq \mathbf{0}$; hence, it is not possible to prove $E \vdash \mathbf{0} + \mathbf{0} = \mathbf{0}$, as expected, because these two terms have a completely different semantics. The conditional axioms **R1**–**R3** are about process constants. Note that **R2** requires that p is not (equal to) $\mathbf{0}$ (condition $p \neq \mathbf{0}$). Note also that these conditional axioms are actually a finite collection of axioms, one for each constant definition: since the set $\mathscr{C}$ of process constants is finite, the instances of **R1**–**R3** are finitely many. Finally, we have axioms **P1**–**P3** for parallel composition.

Theorem 3 [20] **(Sound and Complete).** *For every* $p, q \in \mathscr{P}_{CFM}$, $E \vdash p = q$ *if and only if* $p \sim^{\oplus} q$. □

Table 4. Axioms for team equivalence

A1	Associativity	$x + (y + z) = (x + y) + z$	
A2	Commutativity	$x + y = y + x$	
A3	Identity	$x + \mathbf{0} = x$	if $x \neq \mathbf{0}$
A4	Idempotence	$x + x = x$	if $x \neq \mathbf{0}$
R1	Stuck	if $C \doteq \mathbf{0}$, then $C = \mathbf{0} + \mathbf{0}$	
R2	Unfolding	if $C \doteq p \ \wedge \ p \neq \mathbf{0}$, then $C = p$	
R3	Folding	if $C \doteq p\{C/x\} \ \wedge \ q = p\{q/x\}$, then $C = q$	
P1	Associativity	$x \mid (y \mid z) = (x \mid y) \mid z$	
P2	Commutativity	$x \mid y = y \mid x$	
P3	Identity	$x \mid \mathbf{0} = x$	

4 DNI: Distributed Non-interference

4.1 Definition and Compositional Verification

Definition 12 (Distributed Non-Interference (DNI)). *A CFM process p enjoys the distributed non-interference property (DNI, for short) if for each p', p'', reachable from p, and for each $h \in H$, such that $p' \xrightarrow{h} p''$, we have that $p'\backslash H \sim^{\oplus} p''\backslash H$ holds.* □

This intuitive and simple definition is somehow hybrid, because, on the one hand, it refers to reachable states p' and p'' in the LTS semantics, while, on the other hand, it requires that $p'\backslash H \sim^{\oplus} p''\backslash H$, a condition that can be checked on the Petri net semantics. We can reformulate this definition in such a way that it refers only to the Petri net semantics, a reformulation that will be very useful in proving the following theorem.

Definition 13 (DNI on Petri nets). *Given a CFM process p and the FSMs $[\![p]\!]_\emptyset$ and $[\![p\backslash H]\!]_\emptyset$, we say that p satisfies DNI if for each m', m'', reachable from $dec(p)$ in $[\![p]\!]_\emptyset$, and for each $h \in H$, such that $m' \xrightarrow{h} m''$, we have that the two markings $m'\backslash H$ and $m''\backslash H$ of $[\![p\backslash H]\!]_\emptyset$ are team equivalent.* □

Theorem 4. *A process p is not DNI iff there exists $p_i \in dec(p)$ such that p_i is not DNI.*

Proof. If p is not DNI, then there exist m, m' reachable from $dec(p)$, and $h \in H$, such that $m' \xrightarrow{h} m''$, but the two markings $m'\backslash H$ and $m''\backslash H$ of $[\![p\backslash H]\!]_\emptyset$ are not team equivalent. Because of the shape of FSM transitions, $m' \xrightarrow{h} m''$ is a move that must be due to a transition $s \xrightarrow{h} m$, so that $m' = s \oplus \overline{m}$ and $m'' = m \oplus \overline{m}$. Therefore, $m'\backslash H = s\backslash H \oplus \overline{m}\backslash H$ and $m''\backslash H = m\backslash H \oplus \overline{m}\backslash H$. If $m'\backslash H$ is not team equivalent to $m''\backslash H$, then necessarily $s\backslash H \nsim m\backslash H$. Since m is reachable from $dec(p)$, because of the shape of net transitions, there exists $p_i \in dec(p)$ such that s is reachable from p_i. Summing up, if p is not DNI, we have found a $p_i \in dec(p)$ which is not DNI, because p_i can reach s, transition $s \xrightarrow{h} m$ is firable and $s\backslash H \nsim m\backslash H$. The reverse implication is obvious. □

Corollary 1. *A CFM process p is DNI if and only if each $p_i \in dom(dec(p))$ is DNI.*

Proof. The thesis is just the contranominal of Theorem 4. □

Hence, in order to check whether p is DNI, we first compute $dec(p)$ to single out its sequential components; then, we consider only the elements of $dom(dec(p))$, because it is necessary to check each sequential component only once. For instance, if $p = (q_1 \,|\, q_2) \,|\, (q_1 \,|\, q_2)$, then, assuming q_1 and q_2 sequential, $dec(p) = 2 \cdot q_1 \oplus 2 \cdot q_2$, so that $dom(dec(p)) = \{q_1, q_2\}$, and so we have simply to check whether q_1 and q_2 are DNI.

Corollary 2. *If $p \sim^{\oplus} q$ and p is DNI, then also q is DNI.*

Proof. *By Corollary 1, p is DNI if and only if each $p_i \in dom(dec(p))$ is DNI. Since $p \sim^{\oplus} q$, there exists a $\sim$-relating bijection between $dec(p)$ and $dec(q)$. Therefore, the thesis is implied by the following obvious fact: given two sequential CFM processes p_i, q_j such that $p_i \sim q_j$, if p_i is DNI, then q_j is DNI.* □

4.2 Efficient Verification Based on a Structural Characterization

A very efficient DNI verification can be done with the following algorithm. Given the CFM process p, first compute the nets $[\![p]\!]_\emptyset = (S, A, T, dec(p))$ and $[\![p\backslash H]\!]_\emptyset$. Then, compute bisimilarity $\sim$ on the places of the net $[\![p\backslash H]\!]_\emptyset$. Finally, for each $t \in T$ such that $l(t) \in H$, check whether ${}^\bullet t\backslash H$ and $t^\bullet\backslash H$ are bisimilar: if this is the case for all the high-transitions of $[\![p]\!]_\emptyset$, then p is DNI; on the contrary, if for some t the check fails (e.g., because $t^\bullet = \theta$), then p is not DNI. The correctness of this polynomial algorithm follows by the fact that the net $[\![p]\!]_\emptyset$ is dynamically reduced. Its complexity is essentially related to the problem of computing $\sim$ (cf. Remark 1) for $[\![p\backslash H]\!]_\emptyset$ and to give it a suitable adjacency matrix representation in order to check easily, for each high-transition $t \in T$, whether the two relevant places ${}^\bullet t\backslash H$ and $t^\bullet\backslash H$ are related by $\sim$.

4.3 Typing System

In this section we provide a syntactic characterization of DNI by means of a typing proof system, which exploits the axiomatization of team equivalence. Let us first define an auxiliary operator $r(-)$, which takes in input a CFM process p and returns a CFM process p' obtained from p by pruning its high-level actions, so that $sort(p') \subseteq L$. Its definition is outlined in Table 5, where $r(h.p) = \mathbf{0} + \mathbf{0}$ because the pruning of $h.p$ is to be considered as a deadlock place. For instance, consider $C \doteq h.l.C + l.C$; then $r(C) = C'$, where $C' \doteq \mathbf{0} + \mathbf{0} + l.C'$. Similarly, if $D \doteq l.h.D$, then $r(D) = D'$ where $D' \doteq l.(\mathbf{0} + \mathbf{0})$. It is a trivial observation that the net semantics of $r(p)$ is isomorphic to the (reachable part of the) net semantics of $p\backslash H$ for any CFM process p. Moreover, we also define the set of initial actions of p, denoted by $In(p)$, whose definition is outlined in Table 6.

Table 5. Restriction function

$r(\mathbf{0}) = \mathbf{0}$	$r(l.p) = l.r(p)$	$r(h.p) = \mathbf{0} + \mathbf{0}$
$r(p + p') = r(p) + r(p')$	$r(p \mid p') = r(p) \mid r(p')$	$r(C) = C'$ where $C' \doteq r(p)$ if $C \doteq p$

Then we define a typing system on CFM processes such that, if a process p is typed, then p satisfies DNI and, moreover, if p is DNI, then there exists a process p', obtained by possibly reordering its summands via axioms $\mathbf{A_1} - \mathbf{A_4}$, which is typed. The typing system is outlined in Table 7, where we are using a set I of already scanned constants. A process p is typed if $(p, \emptyset) : dni$ is derivable

Table 6. Initial function

$In(\mathbf{0}) = \emptyset$	$In(\mu.p) = \{\mu\}$	$In(p + p') = In(p) \cup In(p')$
$In(p \mid p') = In(p) \cup In(p')$	$In(C) = In(p)$ if $C \doteq p$	

Table 7. Typing proof system

$\dfrac{}{(\mathbf{0},I) : dni}$	$\dfrac{(p,I) : dni, (q,I) : dni}{(p \mid q,I) : dni}$	$\dfrac{(p,I) : dni, (q,I) : dni, In(p+q) \subseteq L}{(p+q,I) : dni}$
$\dfrac{(p,I) : dni}{(l.p,I) : dni}$	$\dfrac{E \vdash p = \mathbf{0}+\mathbf{0}}{(h.p,I) : dni}$	$\dfrac{C \notin I, C \doteq p, (p, I \cup \{C\}) : dni}{(C,I) : dni}$
$\dfrac{C \in I}{(C,I) : dni}$	$\dfrac{\emptyset \neq sort(p) \subseteq H, (p,I) : dni}{(h.p,I) : dni}$	$\dfrac{(p,I) : dni, p \neq \mathbf{0}, (q,I) : dni, E \vdash r(p) = r(q)}{(h.p+q,I) : dni}$

by the rules; this is often simply denoted by $p : dni$. The need for the argument I is clear in the two rules for the constant C: if C has been already scanned (i.e. $C \in I$), then C is typed; otherwise it is necessary to check that its body p is typed, using a set enriched by C (condition $(p, I \cup \{C\}) : dni$).

We are implicitly assuming that the formation rules in this table respect the syntax of CFM; this means that, for instance, the third rule requires also that p and q are actually guarded processes because this is the case in $p + q$. Note that in this rule we are requiring that $In(p + q)$ is a subset of L, so that, as no high-level action is executable by p and q as their first action, no DNI check is required at this level of the syntax.

The interesting cases are the three rules about action prefixing and the rule about the choice operator when a summand starts with a high-level action. The first rule states that if p is typed, then also $l.p$ is typed, for each $l \in L$. The second rule states that if p is a deadlock place (condition $E \vdash p = \mathbf{0} + \mathbf{0}$), then $h.p$ is typed, for each $h \in H$; note that p cannot be $\mathbf{0}$, because $h.\mathbf{0}$ is not secure. The third rule states that if p is a typed term that can perform at least one action, but only high-level actions (condition $\emptyset \neq sort(p) \subseteq H$), then $h.p$ is typed. The rule about the choice operator states that if we prefix a generic typed process p with a high-level action h, then it is necessary that an additional typed summand q is present such that $p\backslash H$ and $q\backslash H$ are team equivalent; this semantic condition is expressed syntactically by requiring that $E \vdash r(p) = r(q)$, thanks to Theorem 3; note that $p \neq \mathbf{0}$, because $h.\mathbf{0} + \mathbf{0}$ is not secure. It is interesting to observe that this rule covers also the case when many summands start with high-level actions. For instance, $h.l.l.\mathbf{0} + (h.l.(h.l.\mathbf{0} + l.\mathbf{0}) + l.l.\mathbf{0})$ is typed because $l.l.\mathbf{0}$ and $h.l.(h.l.\mathbf{0} + l.\mathbf{0}) + l.l.\mathbf{0}$ are typed and $E \vdash r(l.l.\mathbf{0}) = r(h.l.(h.l.\mathbf{0} + l.\mathbf{0}) + l.l.\mathbf{0})$. This strategy is intuitively correct (i.e., it respects DNI) because, by checking

that the subterm $h.l.(h.l.\mathbf{0} + l.\mathbf{0}) + l.l.\mathbf{0}$ is typed/DNI, we can safely ignore the other summand $h.l.l.\mathbf{0}$, as it does not contribute any initial low-visible behavior.

Now we want to prove that the typing system characterizes DNI correctly. To get convinced of this result, consider again $C \doteq h.l.C + l.C$. This process is DNI because, if $C \xrightarrow{h} l.C$, then $C\backslash H \sim (l.C)\backslash H$, which is equivalent to say that $r(C) \sim r(l.C)$. As a matter of fact, $(C, \emptyset) : dni$ holds, because $(h.l.C+l.C, \{C\}) : dni$ holds, because, in turn, $(l.C, \{C\}) : dni$ and $E \vdash r(l.C) = r(l.C)$. On the contrary, $D \doteq l.h.D$ is not DNI because if $h.D \xrightarrow{h} D$, then $h.D\backslash H \not\sim D\backslash H$ as $h.D\backslash H$ is stuck, while $D\backslash H$ can perform l. As a matter of fact, D is not typed: to get $(D, \emptyset) : dni$, we need $(l.h.D, \{D\}) : dni$, which would require $(h.D, \{D\}) : dni$, which is false, as no rule for high-level prefixing is applicable.

Proposition 1. *For each CFM process p, if $p : dni$, then p satisfies DNI.*

Proof. *By induction on the proof of $(p, \emptyset) : dni$.* □

Note that the reverse implication is not alway true, because of the ordering of summands. For instance, $l.\mathbf{0}+h.l.\mathbf{0}$, which is clearly DNI, is not typed. However, $\{\mathbf{A_1} - \mathbf{A_4}\} \vdash l.\mathbf{0} + h.l.\mathbf{0} = h.l.\mathbf{0} + l.\mathbf{0}$, where the process $h.l.\mathbf{0} + l.\mathbf{0}$ is typed.

Proposition 2. *For each CFM process p, if p is DNI, then there exists p' such that $\{\mathbf{A_1} - \mathbf{A_4}\} \vdash p = p'$ and $p' : dni$.*

Proof. *By induction on the structure of p. By considering an additional parameter I of already scanned constants, the base cases are $(\mathbf{0}, I)$ and (C, I) with $C \in I$. Both cases are trivial as these two terms cannot do anything. The only non-trivial inductive case is about summation. Assume that $p_1 + p_2$ is DNI. If $In(p_1 + p_2) \subseteq L$, then both p_1 and p_2 are DNI; by induction, there exist $p_i' : dni$ such that $\{\mathbf{A_1} - \mathbf{A_4}\} \vdash p_i = p_i'$ for $i = 1, 2$; hence, $\{\mathbf{A_1} - \mathbf{A_4}\} \vdash p_1+p_2 = p_1'+p_2'$ and also $p_1'+p_2' : dni$, as required. On the contrary, if there exists $h \in In(p_1+p_2)$, then there exist q_1 and q_2 such that $\{\mathbf{A_1} - \mathbf{A_4}\} \vdash p_1 + p_2 = h.q_1 + q_2$. Since also $h.q_1 + q_2$ is DNI by Corollary 2, it is necessary that $q_1\backslash H \sim q_2\backslash H$, which is equivalent to $r(q_1) \sim r(q_2)$, in turn equivalent to stating that $E \vdash r(q_1) = r(q_2)$. Moreover, the DNI property has to be satisfied by q_1 and q_2. Hence, by induction, there exist q_1', q_2' such that $\{\mathbf{A_1} - \mathbf{A_4}\} \vdash q_i = q_i'$ and $q_i' : dni$, for $i = 1, 2$. By transitivity and substitutivity, we have $\{\mathbf{A_1} - \mathbf{A_4}\} \vdash p_1 + p_2 = h.q_1' + q_2'$. Moreover, since $E \vdash r(q_1) = r(q_2)$, we also have that $E \vdash r(q_1') = r(q_2')$, and so, by the proof system, $h.q_1' + q_2' : dni$, as required.* □

5 Conclusion

Related Literature. The non-interference problem in a distributed model of computation was first addressed in [5,6]. There, the Petri net class of *unlabeled elementary net systems* (i.e., 1-safe, contact-free nets) was used to describe some information flow security properties, notably *BNDC* (Bisimulation Non-Deducibility on Composition) and *SBNDC* (Strong BNDC), based on interleaving bisimilarity. These two properties do coincide on unlabeled elementary net

systems, but actually SBNDC is stronger on *labeled* FSMs; for instance, the CFM process $l.h.l.\mathbf{0} + l.\mathbf{0} + l.l.\mathbf{0}$ is BNDC [10], while it is not SBNDC; this explains why we have chosen SBNDC as our starting point towards the formulation of DNI.

In [6] it is shown that BNDC can be characterized as a structural property of the net concerning two special classes of places: *causal places*, i.e., places for which there are an incoming high transition and an outgoing low transition; and *conflict places*, i.e. places for which there are both low and high outgoing transitions. The main theorem in [6] states that if places of these types are not present or cannot be reached from the initial marking, then the net is BNDC. An algorithm following this definition was implemented in [11], but it suffers from the high complexity of the reachability problem.

Starting from [6], Baldan and Carraro in [1] provide a *causal characterization* of BNDC on safe Petri nets (with injective labeling), in terms of the unfolding semantics, resulting in an algorithm much better than [11]. Nonetheless, the BNDC property is based on an interleaving semantics and the true-concurrency semantics is used only to provide efficient algorithms to check the possible presence of interferences.

Another paper studying non-interference over a distributed model is [7]. Bérard et al. study a form of non-interference similar to SNNI [10] for *High-level Message Sequence Charts* (HMSC), a scenario language for the description of distributed systems, based on composition of partial orders. The model allows for unbounded parallelism and the observable semantics they use is interleaving and linear-time (i.e., language-based). They prove that non-interference is undecidable in general, while it becomes decidable for regular behaviors, or for weaker variants based on observing local behavior only. Also in this case, however, the truly-concurrent semantics based on partial orders is used mainly for algorithmic purpose; in fact, the authors shows that their decidable properties are PSPACE-complete, with procedures that never compute the interleaving semantics of the original HMSC.

The use of causal semantics in security analysis was advocated also in [12], where Fröschle suggests that for modeling, verification, decidability and complexity reasons (i.e., mainly for algorithmic convenience), this kind of equivalences should be used.

On the contrary, Baldan et al. in [2] define security policies, similar to non-interference, where causality is used as a first-class concept. So, their notion of non-interference is more restrictive than those based on interleaving semantics. However, their approach is linear-time, while non-interference is usually defined on a branching-time semantics, i.e., on bisimulation. Moreover, it seems overly restrictive; for instance, the CFM process $h.l.\mathbf{0} + l.\mathbf{0}$, which is DNI, would be considered insecure in their approach.

Future Research. Our proposal of DNI is customized for CFM and for the simple subclass of Petri nets called *finite-state machines*. However, we think that for finite Place/Transition Petri nets [31] (and its corresponding process algebra FNM [18]), the non-interference property DNI can be formulated as follows:

> Given a finite P/T Petri net N, a marking m is DNI if, for each m', m'' reachable from m, and $\forall h \in H$ such that $m' \xrightarrow{h} m''$, we have that $m'\backslash H \sim_{sfc} m''\backslash H$, where $\sim_{sfc}$ denotes state-sensitive fc-bisimulation equivalence [22].

We may wonder if such a property is decidable. We conjecture that for 1-safe nets, DNI is decidable because *history-preserving bisimilarity* is decidable [33] on this class of nets and $\sim_{sfc}$ is very similar to it. About unbounded P/T nets, we conjecture that DNI is decidable for BPP nets (i.e., nets with singleton preset transitions) because $\sim_{sfc}$ is characterizable as a decidable team-like equivalence [22]. However, for general P/T nets, the problem is less clear. On the one hand, $\sim_{sfc}$ (as any other bisimulation-based behavioral equivalence) is undecidable for *labeled* finite P/T nets with at least two *unbounded* places [9,25]; so, we would expect that DNI is undecidable on this class of nets. On the other hand, for unbounded *partially observed* finite P/T nets (i.e., net with unobservable high transitions and *injective labeling* on low transitions), Best et al. proved in [3] that SBNDC is decidable; so, this positive result gives some hope that it might be possible to prove decidability of DNI, at least on this restricted class of unbounded finite nets.

We plan to extend this approach to a setting with silent transitions as well as to intransitive non-interference [2,3,7,16].

Why True Concurrency? To conclude this paper, we want to recapitulate some further reasons for preferring truly-concurrent semantics over interleaving semantics. A first observation was that interleaving semantics are not a congruence for the action refinement operator [8,13,15], nor for (some forms of) multi-party synchronization (see Chapter 6 of [17]), while this property holds for some truly concurrent semantics. One further (and stronger) observation is about expressiveness. In [18] six process algebras are compared w.r.t. an interleaving semantics and w.r.t. a concrete truly-concurrent semantics based on Petri nets. The resulting two hierachies are quite different; in particular, the hierarchy based on interleaving semantics equates process algebras that have very different expressive power in terms of classes of solvable problems in distributed computing. As it may happen that two Turing-complete process algebras can solve different classes of problems, classic Turing (or sequential) computability, based on computable functions on natural numbers, needs to be extended to deal with computable objects in a distributed model such as Petri nets. This observation calls for the study of a generalization of Turing computability, we call *distributed computability* [21].

Acknowledgments. The anonymous referees are thanked for their comments.

References

1. Baldan, P., Carraro, A.: A causal view on non-intereference. Fundam. Infor. **140**(1), 1–38 (2015)
2. Baldan, P., Beggiato, A., Lluch Lafuente, A.: Many-to-many information flow policies. In: Jacquet, J.-M., Massink, M. (eds.) COORDINATION 2017. LNCS, vol. 10319, pp. 159–177. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59746-1_9
3. Best, E., Darondeau, Ph., Gorrieri, R.: On the decidability of non-interference over unbounded Petri nets. In: Proceedings of the 8th International Workshop on Security Issues in Concurrency (SecCo 2010), EPTCS, vol. 51, pp. 16–33 (2010)
4. Best, E., Devillers, R., Kiehn, A., Pomello, L.: Concurrent bisimulations in Petri nets. Acta Informatica **28**(3), 231–264 (1991). https://doi.org/10.1007/BF01178506
5. Busi, N., Gorrieri, R.: A survey on non-interference with Petri nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 328–344. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27755-2_8
6. Busi, N., Gorrieri, R.: Structural non-interference in elementary and trace nets. Math. Struct. Comput. Sci. **19**(6), 1065–1090 (2009)
7. Bérard, B., Hélouët, L., Mullins, J.: Non-interference in partial order models. In: Proceedings of the ACSD 2015, pp. 80–89. IEEE Computer Society (2015)
8. Castellano, L., De Michelis, G., Pomello, L.: Concurrency versus interleaving: an instructive example. Bull. EATCS **31**, 12–14 (1987)
9. Esparza, J.: Decidability and complexity of petri net problems—an introduction. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_20
10. Focardi, R., Gorrieri, R.: Classification of security properties. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 331–396. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45608-2_6
11. Frau, S., Gorrieri, R., Ferigato, C.: Petri net security checker: structural non-interference at work. In: Degano, P., Guttman, J., Martinelli, F. (eds.) FAST 2008. LNCS, vol. 5491, pp. 210–225. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01465-9_14
12. Fröschle, S.: Causality, behavioural equivalences, and the security of cyberphysical systems. In: [30], pp. 83–98 (2015)
13. van Glabbeek, R., Goltz, U.: Equivalence notions for concurrent systems and refinement of actions. In: Kreczmar, A., Mirkowska, G. (eds.) MFCS 1989. LNCS, vol. 379, pp. 237–248. Springer, Heidelberg (1989). https://doi.org/10.1007/3-540-51486-4_71
14. van Glabbeek, R.J.: Structure preserving bisimilarity: supporting an operational Petri net semantics of CCSP. In: [30], pp. 99–130. Springer (2015)
15. Gorrieri, R., Rensink, A.: Action Refinement. In: Handbook of Process Algebra, pp. 1047–1147. North-Holland (2001)
16. Gorrieri, R., Vernali, M.: On intransitive non-interference in some models of concurrency. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2011. LNCS, vol. 6858, pp. 125–151. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23082-0_5
17. Gorrieri, R., Versari, C.: Introduction to Concurrency Theory: Transition Systems and CCS. EATCS Texts in Theoretical Computer Science. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21491-7

18. Gorrieri, R.: Process Algebras for Petri Nets: The Alphabetization of Distributed Systems. EATCS Monographs in Theoretical Computer Science. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55559-1
19. Gorrieri, R.: Verification of finite-state machines: a distributed approach. J. Log. Algebraic Methods Program. **96**, 65–80 (2018)
20. Gorrieri, R.: Axiomatizing team equivalence for finite-state machines. In: Alvim, M.S., Chatzikokolakis, K., Olarte, C., Valencia, F. (eds.) The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy. LNCS, vol. 11760, pp. 14–32. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31175-9_2
21. Gorrieri, R.: Toward distributed computability theory. In: Reisig, W., Rozenberg, G. (eds.) Carl Adam Petri: Ideas, Personality, Impact, pp. 141–146. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-96154-5_18
22. Gorrieri, R.: A study on team bisimulations for BPP nets (extended abstract). In: Janicki, R., et al. (eds.) Petri Nets 2020. LNCS, vol. 12152, pp. xx–yy. Springer, Cham (2020)
23. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall International Series in Computer Science (1985)
24. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation, 2nd edn. Addison-Wesley, Boston (2001)
25. Jančar, P.: Undecidability of bisimilarity for Petri nets and some related problems. Theor. Comput. Sci. **148**(2), 281–301 (1995)
26. Keller, R.: Formal verification of parallel programs. Commun. ACM **19**(7), 561–572 (1976)
27. Milner, R.: Communication and Concurrency. Prentice-Hall, Upper Saddle River (1989)
28. Nielsen, M., Thiagarajan, P.S.: Degrees of non-determinism and concurrency: a Petri net view. In: Joseph, M., Shyamasundar, R. (eds.) FSTTCS 1984. LNCS, vol. 181, pp. 89–117. Springer, Heidelberg (1984). https://doi.org/10.1007/3-540-13883-8_66
29. Meyer, R., Platzer, A., Wehrheim, H. (eds.): Correct System Design-Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday. LNCS, vol. 9360. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23506-6
30. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987)
31. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, Upper Saddle River (1981)
32. Ryan, P.Y.A.: Mathematical models of computer security. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 1–62. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45608-2_1
33. Vogler, W.: Deciding history preserving bisimilarity. In: Albert, J.L., Monien, B., Artalejo, M.R. (eds.) ICALP 1991. LNCS, vol. 510, pp. 495–505. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-54233-7_158