Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux
Controller Synthesis for Non-Interference Properties.
Technical Report RI-2008-1, April 2008. Available at www.irccyn.fr/franck
IRCCyN/CNRS, 1 rue de la No, BP 92101, 44321 Nantes Cedex, France.

Controller Synthesis for Non-Interference Properties*

Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux

IRCCyN/CNRS
BP 92101
1 rue de la Noë
44321 Nantes Cedex 3
France

Abstract. In this paper, we focus on the synthesis of secure systems. We assume the system is composed of two users, the low level and the high level users. The security property the system must satisfy is a *non-interference* property. A system is *non-interferent* if the low level user cannot deduce any information about the system by playing its low level actions. Various notions of non-interference have been defined in the literature, and in this paper we consider the most popular ones: trace-based non-interference (SNNI) and (bi)simulation-based non-interference (CSNNI and BSNNI). For each of these notions, we study the corresponding synthesis problem, *i.e.*, build a controller s.t. the controlled system is non-interferent.

Key words: Non-Interference, Simulation, Bisimulation, Controller Synthesis.

1 Introduction

Computing environments now allow users to use programs that are sent or fetched from different sites to achieve their goal, either in private or in an organization. Such programs may deal with secret information such as private data (of an user) or as classified data (of an organization). Similar situations may occur in any computing environments where multiple users share common computing resources. One of the basic concerns in such contexts is to ensure programs not to leak sensitive data to a third party, either maliciously or inadvertently. This is what is often called *secrecy*.

In an environment with two parties, *information flow analysis* defines secrecy as: "high level information never flows into low level channels". Such a definition is referred to as a *non-interference* property, and may

* Work supported by the French Government under grant ANR-SETI-003.

capture any causal dependency between high level and low level behaviors. Such properties (namely[1] SNNI, CSNNI, BSNNI) are out of the scope of the common safety/liveness classification of system properties [1]. In recent years, verification of information flow security properties has been a very active domain [1, 2] as it can be applied to the analysis of cryptographic protocols where numerous uniform and concise characterizations of information flow security properties (*e.g.* confidentiality, authentication, non-repudiation or anonymity) in terms of non-interference have been proposed.

In this paper, we consider the problem of *synthesizing* non-interferent systems. The *verification* problem for a given system S and a specification ϕ consists in checking whether S satisfies ϕ which is often written $S \models \phi$ and referred to as the *model-checking problem*. In contrast, the *control problem* assumes the system is *open i.e.*, we can restrict the behavior of S: some events in S are *controllable* (say Σ_c) and the others are *uncontrollable* (Σ_u), and we can disable controllable actions. Given a controller C, the *supervised* system $C(S)$ (read "S supervised by C") is composed of the subset of the behaviors of S that is allowed by C. The *control problem* for a system S and a specification ϕ asks the following: Is there a controller C s.t. $C(S) \models \phi$? The associated *controller synthesis problem* asks to compute a witness controller C.

For security properties such as non-interference, we are given a system S, and a type of non-interference $\phi \in \{\text{SNNI}, \text{CSNNI}, \text{BSNNI}\}$. The ϕ-non-interference verification problem asks whether S is non-interferent for ϕ (we use the shortcut "S is ϕ" in the sequel). The ϕ-non-interference control problem asks the following: Is there a controller C such that $C(S)$ has the ϕ-non-interference property? Moreover we are interested in computing the *most permissive controller* for S in case there is one.

In [3] the authors consider the complexity of many non-interference *verification* problems but control is not considered. There is also a large body of work on the use of static analysis techniques to enforce information flow policies. A general overview may be found in [4]. The non-interference control problem was first considered in [5] for dense-time systems given by timed automata. The non-interference property that was considered is the *state* non-interference property, which is less demanding than the ones we consider here.

This paper is a followup paper of our previous work [6] about *non-interference control problems*. In this previous paper, we have assumed that high level actions (Σ_h) are controllable ($\Sigma_c = \Sigma_h$) and low level ac-

[1] The meaning of these acronyms is defined in Section 3.2

tions (Σ_l) are uncontrollable ($\Sigma_u = \Sigma_l$) *i.e.*, we have studied a particular instance of the control problem. In the present paper we release this assumption and only assume that[2] $\Sigma_h \cup \Sigma_l = \Sigma_c \cup \Sigma_u$. This generalization is interesting in practice because it enables one to specify that an action from Σ_h cannot be disabled (a service must be given), while some actions of Σ_u can be disabled. We can view actions of Σ_u as capabilities of the low level user (*e.g*, pressing a button), and it thus makes sense to prevent the user using the button for instance by disabling it temporarily. It is also of theoretical interest, because these general non-interference control problems are *really* more difficult than the verification problems in the sense that we can *reduce* the verification problems to particular instances of the control problems: it suffices to take $\Sigma_c = \emptyset$ and $\Sigma_u = \Sigma_h \cup \Sigma_l$. This was not the case for the control problems studied in [6]. Moreover, we consider here that the controller has the knowledge of the full history of the system at any point in time. This is also in contrast with [6], where we investigated the SNNI-non-interference control problem in the Ramadge and Wonham setting, *i.e.*, where the knowledge of the controller is the set of actions that have happened from the beginning. The new results of this paper are the following: we reduce the SNNI-controller synthesis problem (with $\Sigma_h \cup \Sigma_l = \Sigma_c \cup \Sigma_u$) to solving a sequence of *safety control problems* and obtain an EXPTIME algorithm in the general case. We also prove that in case a controller exists, there is a most permissive one. We characterize the class of systems for which the SNNI-CSP can be solved in PTIME.

In [6], we proved that the BSNNI-non-interference control problem was decidable (EXPTIME) when $\Sigma_c = \Sigma_h$ and $\Sigma_u = \Sigma_l$. Releasing the assumption raises a lot of issues: for example, we exhibit an example for which there is no most permissive controller.

We then focus on the "easier" CSNNI-non-interference controller synthesis problem. We show that under some assumptions we can reduce it to the SNNI-CSP. Finally, we exhibit an example for which again, no most permissive controller exists.

Organization of the paper. Section 2 recalls the basics of finite automata, languages, bisimulation. Section 3 contains known results about safety control and the definitions of non-interference and the control synthesis problems. Section 4 is the core of the paper and contains the main result: we solve the SNNI-CSP. In Section 5, we show that the BSNNI-CP does not have nice properties (like existence of a most permissive

[2] Still we require $\Sigma_h \cap \Sigma_l = \Sigma_c \cap \Sigma_u = \emptyset$.

controller). Finally, in Section 6 we give a list of open problems and future work.

2 Preliminaries

Let Σ be a finite set, $\varepsilon \notin \Sigma$ and $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$. A *word* w over Σ is a sequence of letters $w = a_0 a_1 \cdots a_n$ s.t. $a_i \in \Sigma$ for $0 \leq i \leq n$. Σ^* is the set of words over Σ. We denote $u.v$ the *concatenation* of two words. As usual ε is also the empty word s.t. $u.\varepsilon = \varepsilon.u = u$. A *language* is a subset of Σ^*. Given a word $w = a_0 a_1 \cdots a_n$ and $L \subseteq \Sigma$ the *projection* of w over L is denoted w/L.

2.1 Labeled Transition Systems

Definition 1 (Labeled Transition System). *A* labeled transition system (LTS) *is a tuple* $\mathcal{S} = (S, s_0, \Sigma^\varepsilon, \rightarrow)$ *where* S *is a set of states,* s_0 *is the initial state,* Σ *a finite alphabet of actions,* $\rightarrow \subseteq S \times \Sigma^\varepsilon \times S$ *is the transition relation. We use the notation* $q \xrightarrow{a} q'$ *if* $(q, a, q') \in \rightarrow$*. A LTS is finite if* S *is finite. We let* $En(q)$ *be the set of labels* a *s.t.* $(q, a, q') \in \rightarrow$ *for some* q'*.* □

A run ρ of $\mathcal{S}$ from s is a finite sequence of transitions $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ s.t. $q_0 = s$ and $(q_i, a_i, q_{i+1}) \in \rightarrow$ for $0 \leq i \leq n-1$. We denote $last(\rho)$ the last state of the sequence *i.e.*, the state q_n, and $q_0 \xrightarrow{*} q_n$ if there is a run from from q_0 to q_n. The set of *reachable* states of $\mathcal{S}$ is $Reach(\mathcal{S}) = \{s \mid s_0 \xrightarrow{*} s\}$. We let $Runs(s, \mathcal{S})$ be the set of runs from s in $\mathcal{S}$ and $Runs(\mathcal{S}) = Runs(s_0, \mathcal{S})$. The *trace* of ρ is $trace(\rho) = a_1 \cdots a_n$.
A *finite automaton* is a tuple $A = (S, s_0, \Sigma^\varepsilon, \rightarrow, F)$ where $(S, s_0, \Sigma^\varepsilon, \rightarrow)$ is a finite LTS and $F \subseteq Q$ is a set of *final* states.

A word $w \in \Sigma^*$ is *generated* by A if $w = trace(\rho)$ for some $\rho \in Runs(A)$ with $last(\rho) \in F$. The language generated[3] by A, $\mathcal{L}(A)$, is the set of words generated by A. Two automata A and B are *language equivalent* denoted $A \approx_{\mathcal{L}} B$ if $\mathcal{L}(A) = \mathcal{L}(B)$ *i.e.*, they generate the same set of words.

2.2 Simulation and Bisimulation

Definition 2 (Weak Simulation). *Let* $A_1 = (S_1, s_0^1, \Sigma^\varepsilon, \rightarrow_1)$*,* $A_2 = (S_2, s_0^2, \Sigma^\varepsilon, \rightarrow)$ *be two LTS. Let* $\mathcal{R} \subseteq S_1 \times S_2$ *be a relation.* $\mathcal{R}$ *is a* weak simulation relation *of* A_2 *by* A_1 *if:*

[3] Notice that we often use *prefix closed* languages and when we omit the set of final states, we implicitly mean that $F = Q$ *i.e.*, all states are final.

1. $s_0^1 \mathcal{R} s_0^2$,
2. *for each* $s \in S_1$, $p \in S_2$ *s.t.* $s\mathcal{R}p$:
   - *if* $p \xrightarrow{\varepsilon^*}_2 p'$ *then* $\exists s'$ *s.t.* $s \xrightarrow{\varepsilon^*}_1 s'$ *and* $s'\mathcal{R}p'$,
   - *if* $p \xrightarrow{\varepsilon^* a \varepsilon^*}_2 p'$ *then* $\exists s'$ *s.t.* $s \xrightarrow{\varepsilon^* a \varepsilon^*}_1 s'$ *and* $s'\mathcal{R}p'$.

A_1 weakly simulates A_2 *if there is a weak simulation of* A_2 *by* A_1. *We write* $A_1 \sqsubseteq_{\mathcal{W}} A_2$ *if* A_1 *weakly simulates* A_2. □

Definition 3 (Weak Bisimulation). *Let* $A_1 = (S_1, s_0^1, \Sigma^\varepsilon, \rightarrow_1)$, $A_2 = (S_2, s_0^2, \Sigma^\varepsilon, \rightarrow)$ *be two LTS.* A_1 *and* A_2 *are* weakly bisimilar *if there is a weak simulation* $\sqsubseteq_{\mathcal{W}}$ *s.t.* $A_1 \sqsubseteq_{\mathcal{W}} A_2$ *and* $A_2 \sqsubseteq_{\mathcal{W}}^{-1} A_1$ *(*i.e., $\sqsubseteq_{\mathcal{W}}$ *and* $\sqsubseteq_{\mathcal{W}}^{-1}$ *are weak simulation relations). We write* $A_1 \approx_{\mathcal{W}} A_2$ *if* A_1 *and* A_2 *are weakly bisimilar.* □

2.3 Restriction and Abstraction

The *abstracted* transition system hides a set of labels $L \subseteq \Sigma$:

Definition 4 (Abstracted Transition System). *Given a LTS* $A = (S, s_0, \Sigma^\varepsilon, \rightarrow)$ *and* $L \subseteq \Sigma$ *we define the LTS* $A/L = (S, s_0, (\Sigma \backslash L)^\varepsilon, \rightarrow_L)$ *where* $q \xrightarrow{a}_L q' \iff q \xrightarrow{a} q'$ *for* $a \in \Sigma \setminus L$ *and* $q \xrightarrow{\varepsilon}_L q' \iff q \xrightarrow{a} q'$ *for* $a \in L \cup \{\varepsilon\}$. □

The *restricted* transition system cuts transitions labeled by the letters in $L \subseteq \Sigma$:

Definition 5 (Restricted Transition System). *Given a LTS* $A = (S, s_0, \Sigma^\varepsilon, \rightarrow)$ *and* $L \subseteq \Sigma$, $A \setminus L$ *is the LTS* $(S, s_0, (\Sigma \setminus L)^\varepsilon, \rightarrow_L)$ *where* $q \xrightarrow{a}_L q' \iff q \xrightarrow{a} q'$ *for* $a \in \Sigma^\varepsilon \setminus L$. □

3 Control and Non-Interference

3.1 Known Results about Safety Control

Let $\mathcal{S} = (S, s_0, \Sigma, \rightarrow)$ be a LTS s.t. the set of actions is partitioned into Σ_u (*uncontrollable actions*) and Σ_c (*controllable actions*). In this case we say A is a *Game* LTS (GLTS).

Definition 6 (Controller). *A* controller C *for* $\mathcal{S}$ *is a partial mapping* $C : \mathit{Runs}(s_0, \mathcal{S}) \rightarrow \Sigma_c$. □

Given $\mathcal{S}$ and C, the set of runs $\mathit{Outcome}(C(\mathcal{S})) \subseteq \mathit{Runs}(s_0, \mathcal{S})$ of the controlled system $C(\mathcal{S})$ is inductively defined by:

– $s_0 \in \mathit{Outcome}(C(\mathcal{S}))$,
– if $\rho \in \mathit{Outcome}(C(\mathcal{S}))$ then $\rho \xrightarrow{l} q' \in \mathit{Outcome}(C(\mathcal{S}))$ if $\mathit{last}(\rho) \xrightarrow{l} q'$ and either $l \in \Sigma_u$, or $l \in C(\rho)$. □

A state q is reachable in $C(\mathcal{S})$ if there is a run $\rho \in \mathit{Outcome}(C(\mathcal{S}))$ s.t. $\mathit{last}(\rho) = q$. $\mathit{Reach}(C(\mathcal{S}))$ denotes the set of reachable states of $C(\mathcal{S})$.

Controllable Predecessors. Let $\mathit{Pre}^l(X) = \{q \mid \exists q' \in X : q \xrightarrow{l} q'\}$. We define $\mathit{Cpre}(X) = \{q \mid \exists l \in \Sigma_c : q \in \mathit{Pre}^l(X) \setminus \mathit{Pre}^l(\bar{X})\}$ and $\mathit{Upre}(X) = \{q \mid \exists l \in \Sigma_u : q \in \mathit{Pre}^l(X)\}$. We let $\pi(X) = (X \cup \mathit{Cpre}(X)) \setminus \mathit{Upre}(\overline{X})$ be the set of controllable predecessors of a set X.

Remark 1. The usual definition of π is $\pi(X) = \mathit{Cpre}(X) \setminus \mathit{Upre}(\overline{X})$; it requires the system to make a controllable move in order to win. In our setting, we accept the controller which prevents all controllable actions. This is why we use this extended version of π. If one wants to impose that a controllable is taken to win (*i.e.*, use the usual definition of π), the results we obtain in the sequel are still valid.

Algorithm for Safety Control. The control objective we are going to use in the sequel are safety objectives. They can be given by a designated set of bad states we want to avoid. Let $A = (S, s_0, \Sigma, \rightarrow, \mathit{Bad})$ be a finite automaton. The *safety control problem* for A is the following:

$$\textit{Is there a controller } C \textit{ s.t. } \mathit{Reach}(C(A)) \cap \mathit{Bad} = \emptyset \ ? \qquad \text{(SafCP)}$$

The classical results (*e.g*, [7]) for (SafCP) are the following:

– it can be solved in linear time in the size of A by computing the greatest fix-point of $X = \pi(X) \cap \overline{\mathit{Bad}}$ (with $\overline{\mathit{Bad}} = S \setminus \mathit{Bad}$). This gives the set $\mathcal{W}$ of winning states of the game;
– if $q_0 \in \mathcal{W}$, there is a controller C s.t. $\mathit{Reach}(C(A)) \cap \mathit{Bad} = \emptyset$. Moreover, there is even a memoryless most permissive controller $\mathcal{C} : S \rightarrow 2^{\Sigma_c}$ defined by: $c \in \mathcal{C}(q)$ if for each q' s.t. $q \xrightarrow{c} q'$, $q' \in \mathcal{W}$.

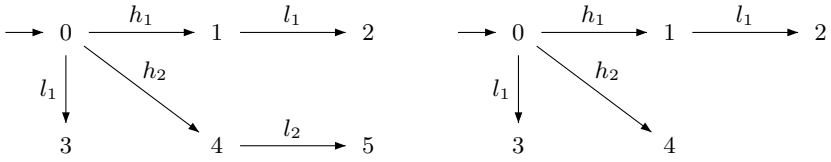
A controller f is memoryless if whenever $\mathit{last}(\rho) = \mathit{last}(\rho')$, $f(\rho) = f(\rho')$. For finite automata, this means that such a controller can be defined as a mapping from the set of states to the set of controllable actions.

3.2 Non-Interference Problems

The *strong non-deterministic non-interference* (SNNI) property has been first proposed by Focardi [1] as a *trace-based* generalization of non-interference for concurrent systems. Let $A = (S, s_0, \Sigma, \rightarrow)$ be a LTS, s.t. $\Sigma = \Sigma_h \cup \Sigma_l$: Σ_l (resp. Σ_h) is the set of *public* (resp. *private*) actions.

Definition 7 (SNNI). *A has the* strong non-deterministic non-interference *property (in short "A is SNNI") if* $A/\Sigma_h \approx_{\mathcal{L}} A\backslash\Sigma_h$. □

Example 1 (SNNI). Figure 1.(a) gives an example of a system which is not SNNI. The high level actions are $\Sigma_h = \{h_1, h_2\}$ and the low level actions are $\Sigma_l = \{l_1, l_2\}$. l_2 is a trace of $\mathcal{A}/\Sigma_h$ but not of $\mathcal{A}\backslash\Sigma_h$ and $\mathcal{A}$ is not SNNI. If we consider $\mathcal{A}$ without the action l_2, $\mathcal{A}$ is SNNI. ■



(a) The automaton $\mathcal{A}$

(b) The Most Liberal Controllable Subgraph of $\mathcal{A}$

Fig. 1. Automaton and Controller

We now give the simulation-based definition of strong non-deterministic non-interference proposed in [1, 5]. Actually, any (bi)simulation-based information flow property presented in [1] could be recast in a similar manner.

Definition 8 (Cosimulation-based SNNI). *A has the* cosimulation-based strong non-deterministic non-interference *property ("A is CSNNI") if* $A\backslash\Sigma_h \sqsubseteq_{\mathcal{W}} A/\Sigma_h$. □

Definition 9 (Bisimulation-based SNNI). *A has the* bisimulation-based strong non-deterministic non-interference *property ("A is BSNNI") if* $A/\Sigma_h \approx_{\mathcal{W}} A\backslash\Sigma_h$. □

The SNNI (resp. CSNNI, BSNNI) *verification problem* (SNNI-VP, resp. CSNNI-VP, BSNNI-VP) is the following: given a LTS A with $\Sigma = \Sigma_h \cup \Sigma_l$, decide whether A is SNNI (resp. CSNNI, BSNNI).

The problem of establishing language equivalence of non-deterministic finite automata is reducible in polynomial time to the problem of checking trace equivalence. Such a problem is known to be PSPACE-complete [8]. Also, SNNI-VP is in PSPACE as well [1]. It is not hard to prove that the SNNI-VP is PSPACE-complete:

Theorem 1. *SNNI-VP is PSPACE-complete.*

Proof. The proof is a reduction of the language equivalence problem for finite state automata (which is PSPACE-complete) to SNNI-VP. Let A and B be two finite automata over the alphabet Σ and with initial states s_A and s_B. Build C and C' as follows (see Fig. 2): the initial state of C (resp. C') is s_A (resp. s_B), the alphabet of C and C' is $\Sigma_h \cup \Sigma_l$ with $\Sigma_l = \Sigma$ and $\Sigma_h = \{h\}$. The transitions of C are defined in Fig. 2. C' is

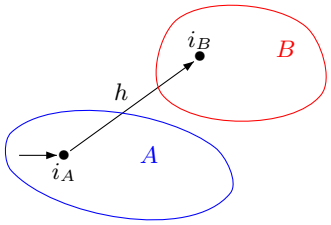


Fig. 2. Reduction of language equivalence to SNNI-VP.

built by swapping the role of A and B in Fig. 2. It is then clear that: $\mathcal{L}(C\backslash\Sigma_h) = \mathcal{L}(A)$ and $\mathcal{L}(C/\Sigma_h) = \mathcal{L}(A) \cup \mathcal{L}(B)$ and $\mathcal{L}(C'\backslash\Sigma_h) = \mathcal{L}(B)$ and $\mathcal{L}(C'/\Sigma_h) = \mathcal{L}(B) \cup \mathcal{L}(A)$. Hence $\mathcal{L}(B) = \mathcal{L}(A)$ iff C and C' have the SNNI property, which can be checked in PSPACE. □

The BSNNI (and CSNNI) verification problems are in PTIME [9].

3.3 Non-Interference Control Problems

The previous non interference problems (SNNI-VP, CSNNI-VP, BSNNI-VP) consist in *checking* whether a LTS has the non-interference property. In case the answer is "no" one has to investigate why the non-interference property does not hold, modify A and check again the property again. In contrast to the verification problem, the control problem indicates whether there is a way of restricting the behavior of users to ensure a given property. Let $\bowtie \in$ {SNNI, CSNNI, BSNNI}. The $\bowtie$-*Control Problem* ($\bowtie$-CP) is the following: given a LTS A and 4 sets $\Sigma_h, \Sigma_l, \Sigma_c, \Sigma_u$ with $\Sigma_l \cap \Sigma_h = \Sigma_c \cap \Sigma_u = \emptyset$ and $\Sigma_h \cup \Sigma_l = \Sigma_c \cup \Sigma_u$,

$$\textit{Is there a controller } C \textit{ s.t. } C(A) \textit{ is } \bowtie? \qquad (\bowtie\text{-CP})$$

Notice that in our previous work [6], we required $\Sigma_h = \Sigma_c$ and $\Sigma_u = \Sigma_l$ and thus the trivial controller that disables Σ_c was always a solution. This is no more true in this more general setting for BSNNI as shown in Section 5. The $\bowtie$-*Control Synthesis Problem* ($\bowtie$-CSP) asks to compute a witness when the answer to the $\bowtie$-CP is "yes", and if there is one, the most permissive controller.

4 SNNI Control Problem

In this section, we show that the SNNI-CSP can be solved by solving iteratively safety control problems.

4.1 An Algorithm for the SNNI-CSP

Let $A = (S, s_0, \Sigma, \rightarrow)$ be a finite automaton with $\Sigma = \Sigma_h \cup \Sigma_l$. Let Σ_c and Σ_u be two disjoint sets s.t. $\Sigma_c \cup \Sigma_u = \Sigma_h \cup \Sigma_l$. If we admit the controller which can disable all the controllable actions, the SNNI-CP can be checked easily:

Theorem 2. *If all the controllable actions can be disabled, the SNNI-CP can be checked in PSPACE.*

Proof. Assume there is a controller C s.t. $C(A)/\Sigma_h = C(A)\backslash\Sigma_h$. We let $C^-(\rho) = C(\rho) \setminus \{c\}$ for $c \in \Sigma_c$. C^- is the same as C but disables c. If $c \in \Sigma_l$, $C^-(A)/\Sigma_h$ and $C^-(A)\backslash\Sigma_h$ obviously generates the same word when restricted to $\Sigma_l \setminus \{l\}$. If $c \in \Sigma_h$, then $C^-(A)$ generates less words than $C(A)$. Hence $\mathcal{L}(C^-(A)/\Sigma_h) \subseteq \mathcal{L}(C(A)/\Sigma_h)$ and as $\mathcal{L}(C^-(A)\backslash\Sigma_h) = \mathcal{L}(C^-(A)\backslash\Sigma_h)$ the result follows[4]. This proves that if there is a controller C s.t. $C(A)$ is SNNI then $A\backslash\Sigma_c$ is SNNI. It then suffices to check that $A\backslash\Sigma_c$ is SNNI to solve SNNI-CP which is a verification problem that can solved in PSPACE. □

In case we do not allow a controller to disable all controllable actions (*i.e.*, we use the more demanding version of the π operator), Theorem 2 does not hold. To solve the SNNI-CP we need to solve the SNNI-CSP. In the sequel we assume $A\backslash\Sigma_h$ is *deterministic*. To compute the most permissive controller, we build a safety game from A and solve a safety control problem. If necessary we iterate this procedure.

Formally, we define $A_2 = (S \cup \{Bad\}, s_0, \Sigma_l, \rightarrow_2)$ with Bad being a fresh state and s.t. A_2 is a *complete* version of $A\backslash\Sigma_h$: $q \xrightarrow{l}_2 q'$ if $q \xrightarrow{l} q'$, and $q \xrightarrow{l}_2 Bad$ if $l \notin En(q)$ and $Bad \xrightarrow{a} Bad$ for $a \in \Sigma_l$.
Because $A\backslash\Sigma_h$ is deterministic, an immediate consequence is:

Fact 1 $w \notin \mathcal{L}(A\backslash\Sigma_h) \iff q_0 \xrightarrow{w}_2 Bad$.

Let $A_1 = A$. Define $A_1 \otimes A_2 = (S \times (S \cup \{Bad\}), (s_0, s_0), \Sigma, \rightarrow)$ by:

- for $h \in \Sigma_h$, $(q_1, q_2) \xrightarrow{h} (q_1', q_2)$ if $q_1 \xrightarrow{h} q_1'$;

[4] Notice that it is always the case that $\mathcal{L}(B\backslash\Sigma_h) \subseteq \mathcal{L}(B/\Sigma_h)$.

– for $l \in \Sigma_l$, $(q_1, q_2) \xrightarrow{l} (q_1', q_2')$ if $q_1 \xrightarrow{l} q_1'$ and $q_2 \xrightarrow{l} q_2'$;

In other words, A_1 mimics A and A_2 records the sequence of events in Σ_l generated by A.

Lemma 1. *Let $w \in \Sigma^*$. $(q_0, q_0) \xrightarrow{w} (q, Bad)$ iff $w/\Sigma_l \notin \mathcal{L}(A \backslash \Sigma_h)$.*

Proof. By definition of $A_1 \otimes A_2$, $(q_0, q_0) \xrightarrow{w} (q, Bad) \iff q_0 \xrightarrow{w/\Sigma_l}_2 Bad$ because A_2 records Σ_l moves. Using Fact 1 $q_0 \xrightarrow{w/\Sigma_l}_2 Bad \iff w/\Sigma_l \notin \mathcal{L}(A \backslash \Sigma_h)$. □

We now state some key theorems about the good controllers of A. Let $Bad^{\otimes} = S \times \{Bad\}$. Given a run $\rho \in Runs(A_1 \otimes A_2)$, we let $\rho_{|1}$ be the projection of ρ on A_1.

Theorem 3. *Let $C : Runs(A) \to \Sigma_c$ be a controller for A satisfying $(H) : \mathcal{L}(C(A)/\Sigma_h) = \mathcal{L}(C(A) \backslash \Sigma_h)$. Let $C^{\otimes}$ be the controller on $A_1 \otimes A_2$ defined by $C^{\otimes}(\rho) = C(\rho_{|1})$. Then, $Reach(C^{\otimes}(A_1 \otimes A_2)) \cap Bad^{\otimes} = \emptyset$.*

Proof. $C^{\otimes}$ is well defined because A_2 is deterministic. Assume $Reach(C^{\otimes}(A_1 \otimes A_2)) \cap Bad^{\otimes} \neq \emptyset$. By definition, there is a run ρ in $Outcome(C^{\otimes}(A_1 \otimes A_2))$ s.t.

$$\rho = (q_0, q_0) \xrightarrow{a_1} (q_1, q_1') \xrightarrow{a_2} \cdots \xrightarrow{a_n} (q_n, q_n') \xrightarrow{l} (q_{n+1}, Bad)$$

Let $\rho_1 = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{q_n} q_n \xrightarrow{l} q_{n+1}$ *i.e.*, ρ_1 is the projection on A_1 of ρ. Let $w = a_1 a_2 \cdots a_n l/\Sigma_l$. We can prove (1): $\rho_1 \in Outcome(C(A))$ and (2): $w \notin \mathcal{L}(C(A) \backslash \Sigma_h)$. (1) directly follows from the definition of $C^{\otimes}$. This implies that $w \in \mathcal{L}(C(A)/\Sigma_h)$. (2) follows from Lemma 1. By (1) and (2) we obtain that $w \in\in \mathcal{L}(C(A)/\Sigma_h) \setminus \mathcal{L}(C(A) \backslash \Sigma_h)$ *i.e.*, $\mathcal{L}(C(A)/\Sigma_h) \neq \mathcal{L}(C(A) \backslash \Sigma_h)$ and so $C(A)$ does not have the SNNI property. This contradicts the assumption (H). Hence $Reach(C^{\otimes}(A_1 \otimes A_2)) \cap Bad^{\otimes} = \emptyset$. □

Theorem 4. *Let $C^{\otimes} : Runs(A_1 \otimes A_2) \to \Sigma_c$ be a controller for $A_1 \otimes A_2$ s.t. $(H') : Reach(C^{\otimes}(A_1 \otimes A_2)) \cap Bad^{\otimes} = \emptyset$. Let $C(\rho) = C^{\otimes}(\rho')$ if $\rho'_{|1} = \rho$. If $C(A) \backslash \Sigma_h = A \backslash \Sigma_h$ then $\mathcal{L}(C(A)/\Sigma_h) = \mathcal{L}(C(A) \backslash \Sigma_h)$.*

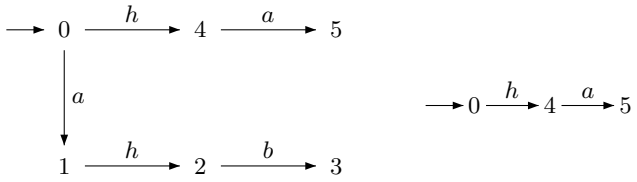
Proof. Let $\rho_1 = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{q_n} q_n$ be a run of A. It is a run of A_1 and as A_2 is deterministic and complete there is exactly one run $\rho = (q_0, q_0) \xrightarrow{a_1} (q_1, q_1') \xrightarrow{a_2} \cdots \xrightarrow{a_n} (q_n, q_n')$ in $A_1 \otimes A_2$ with the same trace $a_1 \cdots a_n$. So C is well defined. Assume $\mathcal{L}(C(A)/\Sigma_h) \neq \mathcal{L}(C(A) \backslash \Sigma_h)$. As $\mathcal{L}(C(A)/\Sigma_h) \supseteq \mathcal{L}(C(A) \backslash \Sigma_h)$ is always satisfied, it must the case that

there is some $w \in \mathcal{L}(C(A)/\Sigma_h) \setminus \mathcal{L}(C(A)\backslash\Sigma_h)$. There is a run $\rho_1 = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ in $C(A)$ s.t. $trace(\rho_1)/\Sigma_l = w$. Let ρ be the unique run with trace $trace(\rho_1) = trace(\rho)$ in $A_1 \otimes A_2$. Write

$$\rho = (q_0, q_0) \xrightarrow{a_1} (q_1, q_1') \xrightarrow{a_2} \cdots \xrightarrow{a_n} (q_n, q_n')$$

First by Lemma 1, $q_n' = Bad$, as $\mathcal{L}(C(A)\backslash\Sigma_h) = \mathcal{L}(A\backslash\Sigma_h)$. Second, this run ρ is in $Outcome(C^{\otimes}(A_1 \otimes A_2))$ because of the definition of C. Hence $Reach(C^{\otimes}(A_1 \otimes A_2)) \cap Bad^{\otimes} \neq \emptyset$ which contradicts (H'). □

If the condition $C(A)\backslash\Sigma_h = A\backslash\Sigma_h$ of Theorem 4 is not satisfied, $C(A)$ is not necessarily SNNI, as we can see in the example of Fig. 3. In this example, $\Sigma_h = \{h\}$ and $\Sigma_c = \{a\}$. We can compute $C(K)$ from $C^{\otimes}$ with $Reach(C^{\otimes}(K_1 \otimes K_2)) \cap Bad^{\otimes} = \emptyset$. Obviously $C(K)$ is not SNNI. For the example of Fig. 1, the condition is true and the most permissive controller is given in Fig. 1.(b). There is an easy sufficient condition for $C(A)$ to be



(a) The automaton K (b) The automaton $C(K)$

Fig. 3. An Automaton K with $C(K)$ not SNNI.

SNNI:

Corollary 1. *If $\Sigma_c \subseteq \Sigma_h$, then $C(A)$ is SNNI.*

Proof. In this case, for any controller $C(A)\backslash\Sigma_h = A\backslash\Sigma_h$ and thus the condition of Theorem 4 is satisfied. □

In this latter case, we can compute the most permissive controller (if there is a solution for the game on $A_1 \otimes A_2$). Let $\tilde{C}$ be defined by $\tilde{C}(\rho) = \mathcal{C}(\rho_{|1})$ with $\mathcal{C}$ the most permissive controller for $A_1 \otimes A_2$. By Theorem 4, $\tilde{C}$ is a good controller and by Theorem 3 it is the most permissive. The example of Fig. 3 shows that computing a most permissive controller on $A_1 \otimes A_2$ does not always suffice. Actually, we may have to iterate the computation of a most permissive controller on the reduced system $C(A)$. Let $\perp$ be

the symbol that denotes non controllability[5] ($\bot$ is the most permissive controller for a system iff it is not controllable). We inductively define a family of controllers and controlled systems as follows:

- let $A^0 = C^0(A) = A$;
- let $\mathcal{D}^{i+1}$ be the most permissive controller for the game $(A_1^i \otimes A_2^i)$ s.t. $\mathit{Reach}(\mathcal{D}^{i+1}(A_1^i \otimes A_2^i)) \cap \mathit{Bad}_i^{\otimes} = \emptyset$ ($\bot$ if no such controller exists). We use the notation $\mathit{Bad}_i^{\otimes}$ because this set depends on A^i. Define $C^{i+1}(\rho) = \mathcal{D}^{i+1}(\rho')$ if $\rho'_{|1} = \rho$. C^{i+1} is well defined because A_2^i is always deterministic. We let $A^{i+1} = C^{i+1}(A)$.

Because of the definition of C^i, C^{i+1} is more restrictive than C^i (it is a controller on a game defined by $C^i(A)$). Hence we can define the controller $\tilde{C}$ by: $\tilde{C} = \cap_{i \geq 0} C^i$ *i.e.*, $c \in \tilde{C}(\rho) \iff c \in C^i(\rho)$ for all $i \geq 0$.

Lemma 2. *Let A' be a subgraph of the unfolding of A and C a controller for A. If $\mathit{Reach}(C^{\otimes}(A'_1 \otimes A'_2)) \cap \mathit{Bad}^{\otimes} \neq \emptyset$ then $\mathcal{L}(C(A)/\Sigma_h) \neq \mathcal{L}(C(A)\backslash\Sigma_h)$.*

Proof. If $\mathit{Reach}(C^{\otimes}(A'_1 \otimes A'_2)) \cap \mathit{Bad}^{\otimes} \neq \emptyset$ there is a run

$$\rho = (q_0, q_0) \xrightarrow{a_1} (q_1, q'_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (q_n, q'_n) \xrightarrow{l} (q_{n+1}, \mathit{Bad})$$

in $C^{\otimes}(A'_1 \otimes A'_2)$. The run $\rho_{|1}$ is a run in $C(A')$ and thus $w = \mathit{trace}(\rho)/\Sigma_l \in C(A')/\Sigma_h$. Thus w is also in $C(A)/\Sigma_h$. Moreover there is no run in $C(A')\backslash\Sigma_h$ that can produce the trace w. Because A'_2 is a subgraph of A_2 and is deterministic, there can be no run in $C(A)\backslash\Sigma_h$ with the trace w. This proves that $\mathcal{L}(C(A)/\Sigma_h) \neq \mathcal{L}(C(A)\backslash\Sigma_h)$. □

Lemma 3. *If C is such that $\mathcal{L}(C(A)/\Sigma_h) = \mathcal{L}(C(A)\backslash\Sigma_h)$, then for each $i \geq 1$, and for each run $\rho \in \mathit{Runs}(A)$, $C(\rho) \in C^i(\rho)$.*

Proof. Assume $C(A)$ is not a subgraph of $C^i(A)$ for some $i \geq 1$. By definition, as C^i is the most permissive controller for A^i, $\mathit{Reach}(C^{\otimes}(A_1^i \otimes A_2^i)) \cap \mathit{Bad}_i^{\otimes} \neq \emptyset$. Applying Lemma 2, as A^i is a subgraph of A, this implies that $\mathcal{L}(C(A)/\Sigma_h) \neq \mathcal{L}(C(A)\backslash\Sigma_h)$ which contradicts the assumption. □

Thus if C is such that $\mathcal{L}(C(A)/\Sigma_h) = \mathcal{L}(C(A)\backslash\Sigma_h)$, then for each run $\rho \in \mathit{Runs}(A)$, $C(\rho) \in \tilde{C}(\rho)$. If we use the more restrictive definition of π *i.e.*, $\pi(X) = \mathit{Cpre}(X) \setminus \mathit{Upre}(\overline{X})$, it may be the case that no controller exists. In this case the following theorem holds:

[5] We want our procedure to be robust *e.g*, if we change the definition of π and impose that a controller must always be able to do a controllable action. This is why we need this $\bot$ controller.

Theorem 5. *If $\tilde{C} = \bot$ then there is no controller C s.t. $\mathcal{L}(C(A)/\Sigma_h) = \mathcal{L}(C(A)\backslash\Sigma_h)$.*

Proof. Direct consequence of Lemma 3. □

The next Lemma is of great importance as it implies that the sequence C^i stabilizes after a finite number of iterations.

Lemma 4. *C^{i+1} is memoryless on $A^i\backslash\Sigma_h$.*

Proof. Recall that memoryless for f means that $f(\rho) = f(\rho')$ whenever $last(\rho) = last(\rho')$. $\mathcal{D}^{i+1}$ is memoryless on $A_1^i \otimes A_2^i$, so it is memoryless on $(A_1^i \otimes A_2^i)\backslash\Sigma_h$. As A_2^i is deterministic, and by definition of the product operator $\otimes, (q_1, q_2) \in ((A_1^i \otimes A_2^i)\backslash\Sigma_h) \Longrightarrow q_1 = q_2$. Let ρ_1 and ρ_2 be two runs in $Runs(A^i\backslash\Sigma_h)$ s.t. $last(\rho_1) = last(\rho_2)$. There exist two runs $\tilde{\rho}^1$ and $\tilde{\rho}^2$ in $Runs((A_1^i \otimes A_2^i)\backslash\Sigma_h)$, s.t. $\tilde{\rho}^1_{|1} = \rho_1$ and $\tilde{\rho}^2_{|1} = \rho_2$. Because $(A_1^i \otimes A_2^i)\backslash\Sigma_h$ is deterministic, we must have $last(\tilde{\rho}^1) = (last(\rho_1), last(\rho_1)) = (last(\rho_2), last(\rho_2)) = last(\tilde{\rho}^2)$. Hence $\mathcal{D}^{i+1}(\tilde{\rho}^1) = \mathcal{D}^{i+1}(\tilde{\rho}^2)$ because $\mathcal{D}^{i+1}$ is memoryless, and by definition $C^{i+1}(\rho_1) = C^{i+1}(\rho_2)$. This proves that C^{i+1} is memoryless on $A^i\backslash\Sigma_h$. □

A consequence of Lemma 4 is that $C^{i+1}(A^i)\backslash\Sigma_h$ is a sub automaton of $C^i(A^{i-1})\backslash\Sigma_h$. It can be computed as follows: let $a_q = C^{i+1}(q, q)$ with (q, q) a state of $C^{i+1}(A^i)\backslash\Sigma_h$. $C^{i+1}(A^i)\backslash\Sigma_h$ is obtained from $C^i(A^i)\backslash\Sigma_h$ by keeping only the transitions in a_q for each state q of $C^i(A^i)\backslash\Sigma_h$ or equivalently by pruning the transitions which are not in a_q. The sequence $C^{i+1}(A^i)\backslash\Sigma_h$ then forms a decreasing sequence and thus stabilizes for some index i_0. C^{i_0} satisfies the condition of Theorem 4.

Theorem 6. *If $\tilde{C} \neq \bot$, then $\mathcal{L}(\tilde{C}(A)/\Sigma_h) = \mathcal{L}(\tilde{C}(A)\backslash\Sigma_h)$.*

Proof. First $\tilde{C} = C^{i_0} \neq \bot$. $Reach(\mathcal{D}^{i_0}(A_1^{i_0} \otimes A_2^{i_0})) \cap Bad^{\otimes} = \emptyset$ by definition of C^{i_0}. Moreover, $C^{i_0}(A^{i_0-1})\backslash\Sigma_h = A^{i_0-1}\backslash\Sigma_h$ and by Theorem 4, $C^{i_0}(A^{i_0-1}) = C^{i_0}(A)$ is SNNI. By Lemma 3, it is also the most permissive controller. □

Remark 2. Although $\tilde{C}$ is the most permissive controller, any sub-controller of $\tilde{C}$ is not necessarily a good controller for SNNI.
For example, let us consider the automaton $\mathcal{B}$ of Fig. 4 with $\Sigma_c = \{a, h\}$. The most permissive controller does not disable anything. Disabling a from state 0 will result in a automaton which is not SNNI. This is in contrast to the usual notion of most permissive controllers where any sub-controller of the most permissive one is a good controller. In the SNNI control problem, the most permissive controller generates the largest controllable subgraph of an automaton which is SNNI.

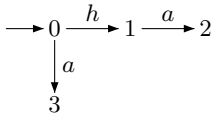


Fig. 4. Automaton $\mathcal{B}$

Theorem 7. *$\tilde{C}$ can be computed in $O(|\rightarrow|.|S|^2)$ steps.*

Proof. Deciding whether there is a controller for $A_1^i \otimes A_2^i$ avoiding $Bad_i^{\otimes}$ can be done in linear time[6] by computing the set of winning states as described in section 3.1. Moreover, if such a controller exists, a memoryless most permissive controller exists and has the size at most $|A_1^i \otimes A_2^i|$. When we compute the first controller C^1 from $C^{\otimes}$, we obtain a controlled system $C^1(A)$ of size $|A|^2$. The important property of this automaton is the following: let (q_1, q_2) and (q_1, q_2') be two states in $C^1(A)_1 \otimes C^1(A)_2$. Because a state q_1 in $C^1(A)$ already contains the state reached in A_2, it must be the case that $q_2 = q_2'$. Hence the controller $C^2(A)$ is memoryless on $C^1(A)$. It can be obtained by cutting transitions in $C^1(A)$ and has size $O(|A|^2)$. From index 2 on, all the controllers are memoryless.

Each time we solve a safety game $C^i(A)_1 \otimes C^i(A)_2$, its size is at most $O(|A|^3)$ (because $C^i(A)_2$ is of size $O(|A|)$). Actually, because the state of A_2^i is already encoded in the state of $A^i 1$ this automaton is a of size $O(|A|^2)$ as well.

The system $C^{i+1}(A), i \geq 1$ can be obtained by pruning in $C^i(A)$ the transitions disabled by C^{i+1}. Hence $C^{i+1}(A)$ is of size $O(|A|^2)$ as well and on the next step we solve again a safety game of size $O(|A|^2)$.

As each time we iterate the computation of a new C^i we remove at least one transition in $C^i(A)\backslash\Sigma_h$: thus we have to solve at most $|\rightarrow|$ safety games of size $O(|A|^2)$. □

Corollary 2. *Let $A\backslash\Sigma_h$ be deterministic. The most permissive controller can be computed in time $O(|S|^4)$ and has size at most $O(|S|^2)$.*

If $A\backslash\Sigma_h$ is not deterministic, we can obtain a deterministic A_2 which has size exponential in $|A|$ and thus the computation of the most permissive controller can de done in exponential time.

4.2 Application

In [1], R. Focardi and R. Gorrieri defined system with two variables that can read and written by two users. We label the corresponding actions

[6] In the size of $A_1^i \otimes A_2^i$.

w_{aij} for writing and r_{aij} for reading, with $a \in \{h, l\}$ is the user (high or low level), $i \in \{0, 1\}$ is the binary variable read or written,, and $j \in \{0, 1\}$ is the value read or written in i. Let $\mathcal{A}_0$ and $\mathcal{A}_1$ be given as in Fig. 5 with w_{lij} and $r_{lij} \in \Sigma_l$ for $i, j \in \{0, 1\}$ and w_{hij} and $r_{hij} \in \Sigma_h$ for $i, j \in \{0, 1\}$. The automaton $\mathcal{A}_i$ represents the effect on the variable i of a read/write action by a user. The synchronized product $\mathcal{A}_0 \times \mathcal{A}_1$ describes the system with the two users updating and reading the variables. $\mathcal{A}_0 \times \mathcal{A}_1$ is not SNNI (which may be hard to infer at first glance ...).

(a) The automaton $\mathcal{A}_0$ (b) The automaton $\mathcal{A}_1$

Fig. 5. $\mathcal{A}_0$ and $\mathcal{A}_1$

We suppose that $\Sigma_u = \{r_{l00}, r_{l01}\}$ and that all other actions are in Σ_c. To make the system SNNI, [1] proposes a monitor $\mathcal{M}$ that restricts the behavior of $\mathcal{A}_0 \times \mathcal{A}_1$ as described by Fig. 6

The solution chosen is to prevent the low level user to read the second variable and the high level user to write in the first variable.

Using our algorithm, we can see that the monitor $\mathcal{M}$ of [1] is not the most permissive one. The controller $\mathcal{C}$ given in Fig. 7 is the most permissive one. It has been computed with UPPAAL-TIGA [10].

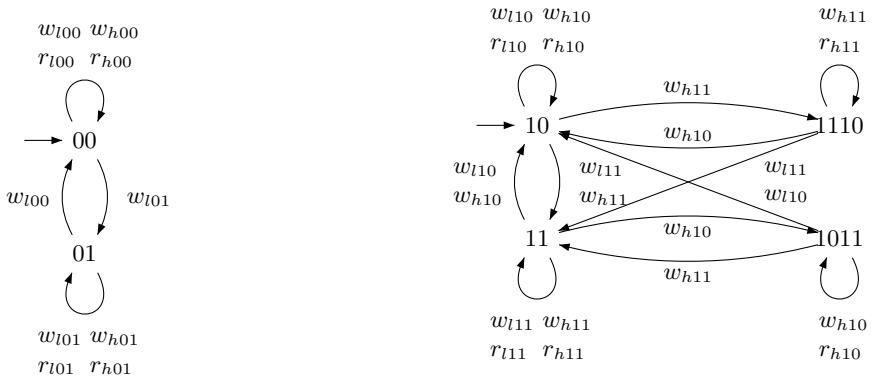
5 BSNNI Control Problem

5.1 Issues with the BSNNI Control Problem

The BSNNI-CP does not have nice properties. First BSNNI is not equivalent to SNNI even if $A \backslash \Sigma_h$ is deterministic as demonstrated by the example $\mathcal{F}$ of Fig. 8(a). We assume $\Sigma_l = \{l_1\}$ and $\Sigma_h = \{h_1\}$.

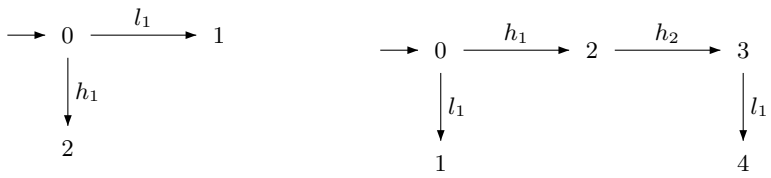
(a) The automaton $\mathcal{M}(\mathcal{A}_0)$ (b) The automaton $\mathcal{M}(\mathcal{A}_1)$

Fig. 6. Monitor $\mathcal{M}$ of [1] for $\mathcal{A}_0$ and $\mathcal{A}_1$



(a) The automaton $\mathcal{C}(\mathcal{A}_0)$ (b) The automaton $\mathcal{C}(\mathcal{A}_1)$

Fig. 7. Most Permissive Controller for $\mathcal{A}_0 \times \mathcal{A}_1$



(a) Automaton $\mathcal{F}$ (b) Automaton $\mathcal{H}$

Fig. 8. Two Automata

This example also demonstrates that sometimes there is no most permissive controller. Assume $\Sigma_u = \emptyset$. $\mathcal{F}$ can be controlled[7] to be BSNNI either by disabling h_1 (controller C_1) or l_1 (controller C_2). Nevertheless neither $C_1(\mathcal{F})$ is a subgraph of $C_2(\mathcal{F})$ nor $C_2(\mathcal{F})$ is a subgraph of $C_1(\mathcal{F})$ and there is no more permissive controller.
Sometimes BSNNI can be enforced, but the most restrictive controller $(A\backslash\Sigma_c)$ does not work. For the automaton $\mathcal{H}$ of Fig. 8(b), with $\Sigma_c = \{h_2\}$ and $\Sigma_u = \{l_1, h_1\}$, we can see that $\mathcal{H}$ is BSNNI, but that $\mathcal{H}\backslash\Sigma_c$ is not. Those problems make the BSNNI control problem a lot harder than the SNNI control problem.

5.2 Issues with the CSNNI Control Problem

As the BSNNI control problem does not have nice properties, we can focus on the easier CSNNI control problem. If $\mathcal{A}\backslash\Sigma_h$ is deterministic, SNNI is equivalent to CSNNI and we can use the results of Section 4.
When $\mathcal{A}\backslash\Sigma_h$ is not deterministic, there is a counterpart of Theorem 2 for CSNNI:

Theorem 8. *If all the controllable actions can be disabled, the CSNNI-CP can be checked in PTIME.*

Proof. The proof of Theorem 2 extends easily to CSNNI. Thus there is a controller C s.t. $C(A)\backslash\Sigma_h \sqsubseteq_{\mathcal{W}} C(A)/\Sigma_h$ iff $A\backslash\Sigma_c$ is CSNNI. This can be checked in PTIME. □

For the CSNNI-CP, there is not always a most permissive controller as the following example on Fig. 9 demonstrates.

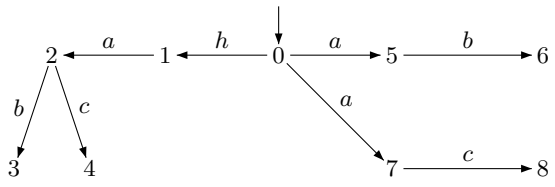


Fig. 9. Automaton $\mathcal{J}$

In this example, $\Sigma_c = \{b, c\}$, $\Sigma_u = \{a, h\}$ and $\Sigma_h = \{h\}$. We have two good memoryless controllers for $\mathcal{J}$: f_1 with $f_1(2) = b$ and f_2 with $f_2(2) = c$ ($f_i(5) = b$ and $f_i(7) = b$). Nevertheless, the "union" controller $f(2) = \{b, c\}$ is not a solution to CSNNI control problem on $\mathcal{J}$.

[7] We assume we can disable all the controllable actions.

There is also no simple condition on the ordering of Σ_c and Σ_h that implies the existence of a most permissive controller for the CSNNI control problem.

6 Conclusion and Future Work

In this paper we have studied non-interference control problems. The results we have obtained are: (1) the SNNI-CP can be solved in EXPTIME (PTIME if $A\backslash\Sigma_h$ is deterministic) by solving a finite sequence of safety games; (2) for the SNNI-CSP, if the system is controllable there is always a most permissive controller; (3) for the BSNNI as well as for the CSNNI control problem, there is not always a most permissive controller which makes those problems harder.
Our future work will consist in:

1. finding the exact complexity of the SNNI-CP (an upper bound is EXPTIME);
2. extend the result of Section 4 to systems given by timed automaton [11]. This can be done for classes of timed automata which are determinizable (and a result of [5] is that the SNNI-CP is undecidable for non-deterministic timed automata);
3. determine conditions under which a most permissive controller exists for the BSNNI-CP and CSNNI-CP. Such conditions should extend the one we gave in [6] *i.e.*, $\Sigma_c = \Sigma_h$.

Acknowledgements

The authors wish to thank Serge Haddad for pointing out a glitch in the preliminary version of the proof of Theorem 7.

References

1. Focardi, R., Gorrieri, R.: Classification of security properties (part I: Information flow). In Focardi, R., Gorrieri, R., eds.: Foundations of Security Analysis and Design I: FOSAD 2000 Tutorial Lectures. Volume 2171 of Lecture Notes in Computer Science., Heidelberg, Springer-Verlag (2001) 331–396
2. Focardi, R., Gorrieri, R.: The compositional security checker: A tool for the verification of information flow security properties. IEEE Trans. Softw. Eng. **23**(9) (1997) 550–571
3. van der Meyden, R., Zhang, C.: Algorithmic verification of noninterference properties. In: Proceedings of the Second International Workshop on Views on Designing Complex Architectures (VODCA 2006). Volume 168 of Electronic Notes in Theoretical Computer Science., Elsevier (2006) 61–75

4. Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications **21**(1) (2003)
5. Gardey, G., Mullins, J., Roux, O.H.: Non-interference control synthesis for security timed automata. In: 3rd International Workshop on Security Issues in Concurrency (SecCo'05). Electronic Notes in Theoretical Computer Science, San Francisco, USA, Elsevier (2005)
6. Cassez, F., Mullins, J., Roux, O.H.: Synthesis of non-interferent systems. In: 4th Int. Conf. on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07). Volume 1 of Communications in Computer and Inform. Science., Copyright Springer (2007) 307–321
7. Thomas, W.: On the synthesis of strategies in infinite games. In: Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95). Volume 900., Springer (1995) 1–13 Invited talk.
8. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time: Preliminary report. In: STOC, ACM (1973) 1–9
9. Bouali, A.: Weak and branching bisimulation in fctool. Research report, INRIA, Sophia-Antipolis, France (1992)
10. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In Damm, W., Hermanns, H., eds.: CAV. Volume 4590 of LNCS., Springer (2007) 121–125
11. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235