Synthesis of Non-Interferent Distributed Systems[⋆]

Franck Cassez[1], John Mullins[2] and Olivier H. Roux[1]

[1] IRCCyN/CNRS
BP 92101 1 rue de la Noë 44321 Nantes Cedex 3 France.
[2] École Polytechnique de Montréal
P.O. Box 6079, Station Centre-ville, Montreal (Quebec), Canada, H3C 3A7.

Abstract. In this paper, we focus on distributed systems subject to security issues. Such systems are usually composed of two entities: a high level user and a low level user that can both do some actions. The security properties we consider are non-interference properties. A system is non-interferent if the low level user cannot deduce any information by playing its low level actions. Various notions of non-interference have been defined in the literature, and in this paper we focus on two of them: one trace-based property (SNNI) and another bisimulation-based property (BSNNI).
For these properties we study the problems of synthesis of a high level user so that the system is non-interferent. We prove that a most permissive high level user can be computed when one exists.

1 Introduction

Security in Distributed Systems. Nowadays computing environments allow users to employ programs that are sent or fetched from different sites to achieve their goal, either in private or in an organization. Such programs may be run as a code to do simple calculation task or as interactive communicating programs doing IO operations or communications. Sometimes they deal with secret information such as a private data of a 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 context is to ensure programs not to leak sensitive data to a third party, either maliciously or inadvertently. This is one of the key aspects of the security concerns, that is often called *secrecy*.

[⋆] Work supported by the French Government under the projects DOTS (*Distributed Open Timed Systems*, `http://www.lsv.ens-cachan.fr/anr-dots/`) sponsored by the ANR and CORTOS (*Control and Observation of Real-Time Open Systems*, `http://www.lsv.ens-cachan.fr/aci-cortos/`)

Non interference. The *information flow analysis* addresses this concern by clarifying conditions when a flow of information in a program is safe (i.e. high level information never flows into low level channels). These conditions referred to as *non interference* properties, capture any causal dependency between high level and low level behaviours. Their characterization has appeared rapidly out of the scope of the common safety/liveness classification of system properties considered by the system verification community during the last twenty five years. Also, in recent years, verification of information flow security has become an emergent field of research in computer science. 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.

Control vs. Verification. 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*. The *control problem* assumes the system is *open i.e.* we can restrict the behaviour of S: some events in S are *controllable* and the others are *uncontrollable*, and we can sometimes disable controllable actions. A *controller* C for S is a mapping which gives, for any history[3] ρ of the system S, the controllable action that can be played (a controller cannot restrict uncontrollable actions). The *supervised* system $S \times C$ (read "S supervised by C") is composed of the subset of the behaviours of S that can be generated using the action prescribed by C. The *control problem* for a system S and a specification ϕ asks the following: Is there a controller C s.t. $S \times C \models \phi$? The associated *control synthesis problem* asks to compute a witness controller C.

Controlling Non-Interference. In this paper we introduce *non-interference control problems*. In this setting we assume that high level actions are controllable and low level actions are uncontrollable. Given a system S, a type of non-interference $\phi \in \{\text{SNNI}, \text{BSNNI}\}$, the ϕ-non-interference control problem asks the following: Is there a controller C such that $S \times C$ has the ϕ-non-interference property? The associated synthesis problems ask to compute a witness controller.

Related Work & Our Contribution. [15] considers the complexity of many non-interference verification problems but control is not considered in this paper. [2] presents a tranformation that removes timing

[3] If the system is a finite automaton, the history is the complete run with states and labels of the transitions.

leaks from programs to make them non-interferent wrt a bisimulation condition. The transformation is decidable but this problem is still different from that of control. Indeed, the transformation removes timing leaks from programs without untimed leaks, by padding with dummy instructions where needed. The non-interference control problem was first considered in [6] for dense timed systems given by timed automata. The timed non interference properties are expressed in terms of states equivalence and co-simulation relations. These control problems are proved decidable and the associated synthesis problems are computable but lead to a non-interferent controlled system which is not necessarily the most permissive.

In this paper, we precisely define the non-interference control problems for two types of non-interference properties: a trace-based property, SNNI and a bisimulation-based property, BSNNI. We show that both problems are decidable. Moreover, given a system S, we prove that a *most permissive* non-interferent (sub)system of S can be computed. To our knowledge, non-interference control problems for finite state systems have not been considered so far.

Organization of the paper. Section 2 recalls the basics of finite automata, languages, bisimulation, and μ-calculus. Section 3 is devoted to the definition of non-interference and known results about control problems for finite automata. Section 4 is the core of the paper and contains the main results: control of SNNI and BSNNI. Section 5 gives the directions of further work.

2 Preliminaries

Let Σ be a finite set and $\varepsilon \notin \Sigma$. We let $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$. 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 two languages A and B over Σ, $A.B$ is the set of words defined $A.B = \{w \in \Sigma^* \mid w = u.v \text{ with } u \in A, v \in B\}$. The set of mappings from A to B is denoted $[A \to B]$. 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* $A = (S, s_0, \Sigma^\varepsilon, \to)$ *where* S *is a set of states,* s_0 *is* *the initial state, Σ a finite alphabet of actions and $\rightarrow \subseteq S \times \Sigma^{\varepsilon} \times S$ is the transition relation. We use the notation $q \xrightarrow{a} q'$ if $(q, a, q') \in \rightarrow$. A LTS is finite is Q 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 A from s is a finite sequence of transitions $\rho = s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ s.t. $(s_i, a_i, s_{i+1}) \in \rightarrow$ for $0 \leq i \leq n-1$. We let $Runs(s, A)$ be the set of runs from s in A and $Runs(A) = Runs(s_0, A)$. The *trace* of ρ is $trace(\rho) = a_0 a_1 \cdots a_n$. A word $w \in \Sigma^*$ is *generated* by A if $w = trace(\rho)$ for some $\rho \in Runs(A)$. The language generated[4] by A, $\mathcal{L}(A)$, is the set of words generated by A.

Two transitions systems 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.

Definition 2 (Product of LTS). *Let $A_1 = (S_1, s_0^1, \Sigma_1, \rightarrow)$ and $A_2 = (S_2, s_0^2, \Sigma_2, \rightarrow)$ be two LTS. The* synchronized product *of A_1 and A_2 is the LTS $A_1 \times A_2 = (S, s_0, \Sigma, \rightarrow)$ defined by: $S = S_1 \times S_2$, $s_0 = (s_0^1, s_0^2)$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $(s_1, s_2) \xrightarrow{\lambda} (s'_1, s'_2)$ if (i) either $s_i \xrightarrow{\lambda} s'_i$ for $i \in \{1, 2\}$ with $\lambda \in \Sigma \setminus \Sigma_{2-i+1}$ and $s'_{2-i+1} = s_{2-i+1}$ or (ii) $s_i \xrightarrow{\lambda} s'_i$ for $i \in \{1, 2\}$ if $\lambda \in \Sigma_1 \cap \Sigma_2$.* □

2.2 Bisimulation, Restriction and Abstraction

Definition 3 (Bisimulation). *Let $A = (S_A, s_0^A, \Sigma^{\varepsilon}, \rightarrow)$, $B = (S_B, s_0^B, \Sigma^{\varepsilon}, \rightarrow)$ be two LTS. $\mathcal{R} \subseteq S_A \times S_B$ is a* bisimulation *if*

1. *for each $s \in S_A$ if $s \xrightarrow{\lambda} s'$, then there is a state $t \in S_B$ s.t. $s \,\mathcal{R}\, t$ and $t \xrightarrow{\lambda} t'$ and $s' \,\mathcal{R}\, t'$;*
2. *for each $t \in S_B$ if $t \xrightarrow{\lambda} t'$, then there is a state $s \in S_A$ s.t. $s \,\mathcal{R}\, t$ and $s \xrightarrow{\lambda} s'$ and $s' \,\mathcal{R}\, t'$.*

Two LTS A and B are strongly bisimilar *if there is a bisimulation $\mathcal{R}$ for A and B s.t. $(s_0^A, s_0^B) \in \mathcal{R}$. We write $A \approx_{\mathcal{S}} B$ if A and B are strongly bisimilar.* □

Definition 4 (ε-Abstract LTS). *Let $A = (S, s_0, \Sigma^{\varepsilon}, \rightarrow)$ be a LTS. The* ε-abstract *LTS associated with A is $A^{\varepsilon} = (S, s_0, \Sigma, \rightarrow_{\varepsilon})$ such that $s \xrightarrow{u}_{\varepsilon} s'$ iff there is a run $s \xrightarrow{\varepsilon^*.u.\varepsilon^*} s'$ in A.* □

[4] Notice that $\mathcal{L}(A)$ is *prefix closed* as we use LTSs that have no accepting or final states.

Two LTS A and B are *weakly bisimilar*, denoted $A \approx_{\mathcal{W}} B$, if $A^\varepsilon \approx_{\mathcal{S}} B^\varepsilon$.

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

Definition 5 (Abstracted Transition system). *Given a LTS $A = (S, s_0, \Sigma^\varepsilon, \rightarrow)$ and $L \subseteq \Sigma$ we define the LTS $A/L = (S', s'_0, \Sigma'^\varepsilon, \rightarrow_L)$ by: $S' = S$, $s'_0 = s_0$, $\Sigma' = \Sigma \setminus L$ and $q \xrightarrow{a}_L q' \iff q \xrightarrow{a} q'$ 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 some letter in $L \subseteq \Sigma$:

Definition 6 (Restricted Transition System). *Given a LTS $A = (S, s_0, \Sigma^\varepsilon, \rightarrow)$ and $L \subseteq \Sigma$, $A \setminus L$ is the LTS $(S', s'_0, \Sigma'^\varepsilon, \rightarrow_L)$ where: $S' = S$, $s'_0 = s_0$, $\Sigma' = \Sigma \setminus L$ and $q \xrightarrow{a}_L q' \iff q \xrightarrow{a} q'$ for $a \in \Sigma^\varepsilon \setminus L$.* □

2.3 The modal μ-calculus & Characteristic Formulæ

The modal μ-calculus was introduced by Kozen [8]. It is used to specify properties of LTS and often called the assembly language w.r.t. temporal logics formalisms. A recent survey on the subject can be found in [3]. Let $\mathcal{V}$ be a countable set of *formula variables* and $A = (S, s_0, \Sigma, \rightarrow)$ be a LTS. We consider the modal μ-calculus with formulas in positive normal form. It is defined by the following grammar:

$$\phi ::= \mathsf{tt} \mid \mathsf{ff} \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [\sigma]\phi \mid \langle\sigma\rangle\phi \mid Z \mid \mu Z.\phi \mid \nu Z.\phi. \qquad (1)$$

where $\sigma \in \Sigma$, $Z \in \mathcal{V}$ and $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ is the set of boolean values. If ϕ is generated by the previous grammar we say that ϕ is a μ-formula or formula for short. A *closed formula* (formula for short in the sequel) of the μ-calculus is a formula s.t. every variable is under the scope of an operator ν or μ. As usual we agree on $\wedge_{\phi \in \emptyset} \phi = \mathsf{tt}$ and $\vee_{\phi \in \emptyset} \phi = \mathsf{ff}$. We denote $[\![\varphi]\!]^A_\rho \subseteq S$ the interpretation of φ w.r.t. a LTS A and a *context* $\rho \in [\mathcal{V} \longrightarrow 2^S]$. $[\![\varphi]\!]^A_\rho$ is defined inductively by the equations of Figure 1. For the semantics definition we use the notations:

- let $Z \in \mathcal{V}$ and ρ be a context, $\xi \in 2^S$, $\rho[Z \mapsto \xi]$ is the context defined by $\rho[Z \mapsto \xi](X) = \rho(X)$ if $X \neq Z$ and $\rho[Z \mapsto \xi](Z) = \xi$;
- given $\sigma \in \Sigma$, $[\![[\sigma]]\!]^A$ and $[\![\langle\sigma\rangle]\!]^A \in [2^S \longrightarrow 2^S]$ are the predicate transformers defined as follows:

$$[\![[\sigma]]\!]^A(X) = \{s \in S : \forall s' \text{ s.t. } s \xrightarrow{\sigma} s' \text{ then } s' \in X\}$$
$$[\![\langle\sigma\rangle]\!]^A(X) = \{s \in S : \exists s' \text{ s.t. } s \xrightarrow{\sigma} s' \text{ and } s' \in X\}.$$

$$
\begin{array}{rcl}
[\![Z]\!]^A_\rho & = & \rho(Z) \\
[\![\neg\phi]\!]^A_\rho & = & S \setminus [\![\phi]\!]^A_\rho \\
[\![\phi \wedge \psi]\!]^A_\rho & = & [\![\phi]\!]_\rho \cap [\![\psi]\!]^A_\rho \\
[\![\phi \vee \psi]\!]^A_\rho & = & [\![\phi]\!]_\rho \cup [\![\psi]\!]^A_\rho \\
[\![\langle\sigma\rangle\phi]\!]^A_\rho & = & [\![\langle\sigma\rangle]\!]^A([\![\phi]\!]^A_\rho) \\
[\![[\sigma]\phi]\!]^A_\rho & = & [\![[\sigma]]\!]^A([\![\phi]\!]^A_\rho) \\
[\![\nu Z.\phi]\!]^A_\rho & = & \bigcup\{\xi \subseteq S : \xi \subseteq [\![\phi]\!]^A_{\rho[Z\mapsto\xi]}\} \\
[\![\mu Z.\phi]\!]^A_\rho & = & \bigcap\{\xi \subseteq S : [\![\phi]\!]^A_{\rho[Z\mapsto\xi]} \subseteq \xi\}.
\end{array}
$$

Fig. 1. Semantics of the modal μ-calculus

For a closed formula ϕ, one has that $[\![\phi]\!]^A_\rho = [\![\phi]\!]^A_{\rho'}$, for any contexts ρ, ρ'. In this case, we simply use $[\![\phi]\!]^A$. We define the *satisfaction relation* $\models$ by $A, s \models \phi$ if and only if $s \in [\![\phi]\!]^A$. If $A, s_0 \models \phi$, we say that A is a model of ϕ, in short $A \models \phi$.

Definition 7. *Let $A = (S, s_0, \Sigma, \rightarrow)$ be a LTS. A formula ϕ s.t. $A \models \phi$ is a* characteristic formula *of A up to strong bisimulation (or simply* characteristic formula *of A) if for any LTS B, $A \approx_{\mathcal{S}} B$ if and only if $B \models \phi$.* □

It can be proved that characteristic formulas exist for finite LTS (see [1, 9] for a detailed presentation) and it is computed as follows. Let $A = (S, s_0, \Sigma, \rightarrow)$. The characteristic formula *CF*(A) of A is given by the system of equations for each $q \in S$:

$$X_q = \bigwedge_{a, q', \, q \xrightarrow{a} q'} \langle a \rangle X_{q'} \wedge \bigwedge_{a \in \Sigma} [a] \Bigg(\bigvee_{q' \in Q, q \xrightarrow{a} q'} X_{q'} \Bigg)$$

By definition of *CF*(A) we have:

Lemma 1. *$A \approx_{\mathcal{S}} B \iff B \models CF(A)$.*

Notice also that characteristic formulas can be derived for weak bisimulation [9].

3 Control and Non-Interference

3.1 Control Problems

Let $A = (S, s_0, \Sigma, \to)$ 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 8 (Controller). *A* controller *for A is a (possibly infinite) LTS $C = (Q, q_0, \Sigma, \to)$ which is* complete *w.r.t. Σ_u: $\forall q \in Q, \forall u \in \Sigma_u$, there is some $q' \in Q$ s.t. $(q, u, q') \in \to$.* $\square$

Supervisory Control Problem [11, 12]. Given a language L, the *prefix closure* of L, denoted $\overline{L}$ is the set of prefixes of words in L. L is a *closed language* if $\overline{L} = L$. Let A be a GLTS and $\emptyset \subset K \subseteq \mathcal{L}(A)$ be a closed language. K is *controllable* w.r.t. $(\mathcal{L}(A), \Sigma_u)$ if $\overline{K}.\Sigma_u \cap \mathcal{L}(A) \subseteq \overline{K}$.

The *supervisory control problem (SCP)* asks the following: given A and K, is there a controller C s.t. $\mathcal{L}(C \times A) = K$?

One result in [11, 12] is that such a controller exists iff K is controllable. In case K is not controllable, one can compute the largest language included in K that is controllable: it is called the *supremal controllable sub-language* of K. This problem is referred to as the *supremal controllable sub-language problem*:

$$\textit{Compute the supremal controllable sub-language of } K. \qquad \text{(SCSLP)}$$

Let $\sup(A, K)$ be the supremal controllable sub-language. By definition any controllable language for A is a subset of $\sup(A, K)$.

Assume K is a language given by a *deterministic* finite automaton A_K. Computing $\sup(A, K)$ can be done in polynomial time in the size of A and A_K. In case $\sup(A, K) \neq \emptyset$ we can even obtain the *most liberal controller* C s.t. $\mathcal{L}(C \times A) = \sup(A, K)$ in polynomial time. C can be represented by a finite state automaton: one can prove (*e.g* [11, 12, 14]) that C is a memoryless [5] controller for a GLTS $A \times \bar{A}_K$ where $\bar{A}_K$ generates the complement language of K. Hence the most liberal controller C is a finite memory[6] controller that has size at most $|A| \cdot |A_K|$.

[5] C is a memoryless controller for A if C can be described using the states of A: the controllable events that are allowed after a particuler sequence of events depend only on the state that is reached in A after reading this sequence of events.

[6] C may need more states than the one of A and in this sense it has finite memory.

μ-Calculus Control Problem [4, 13]. Given a LTS A, a closed μ-formula φ, the *μ-control problem* (μ-CP) is the following:

$$\textit{Is there a controller } C \textit{ such that } C \times A \models \varphi \textit{ ?} \qquad (\mu\text{-CP})$$

This problem has been proved to be solvable in [4, 13]. An algorithm to solve μ-CP is given [4]: 1) first construct a *modal-loop formula*[7] $\varphi(A)$ which is a *quotient automaton* of φ by A s.t. $P \models \varphi(A)$ iff $P \times A \models \varphi$; 2) transform $\varphi(A)$ into a non *deterministic loop automaton* and synthesize a controller for this automaton. This transformation into automaton may cause an exponential blow-up and the complexity of the μ-CP depends on many parameters. The exact complexity of μ-CP is not yet known nevertheless a finite memory most permissive controller can be synthesized if A is controllable w.r.t. φ.

3.2 Non Interference Problems

The *strong non-deterministic non interference* (SNNI) property has been first proposed by Focardi [5] as a *trace-based* generalization of non interference for concurrent systems. Let $A = (S, s_0, \Sigma, \rightarrow)$ be a GLTS, s.t $\Sigma = \Sigma_u \cup \Sigma_c$: Σ_u is the set of *public* actions and Σ_c the set of *private* actions.

Definition 9 (SNNI). *A has the* strong non-deterministic non interference *(SNNI) property if and only if $A/\Sigma_c \approx_{\mathcal{L}} A\backslash\Sigma_c$.* □

The SNNI *verification problem* (SNNI-VP) is the following: given a GLTS A with $\Sigma = \Sigma_u \cup \Sigma_c$, decide whether A has the SNNI property. 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. Also, SNNI-VP is in PSPACE as well [5].

Example 1 (SNNI). Figure 2 gives an example of a system that has not the SNNI property. The high level (controllable) actions are $\Sigma_c = \{h_1, h_2\}$ and the low level (uncontrollable) actions are $\Sigma_u = \{l_1, l_2\}$. l_2 is a trace of A/Σ_c but not of $A\backslash\Sigma_c$ and $\mathcal{A}$ does not have the SNNI property. If we consider A without the action h_2, l_2 (no states 4 and 5) then A satisfies the SNNI property. ■

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

[7] Modal-loop formulas are μ-formulas extended with a modality to check loops.

$$\begin{tikzcd}
{} \arrow[r] & 0 \arrow[r, "h_1"] \arrow[d, "l_1"'] \arrow[dr, "h_2"] & 1 \arrow[r, "l_1"] & 2 \\
& 3 & 4 \arrow[r, "l_2"] & 5
\end{tikzcd}$$

Fig. 2. The automaton $\mathcal{A}$

Definition 10 (Bisimulation-based SNNI). *A has the* bisimulation-based strong non-deterministic non interference *(BSNNI) property if and only if $A/\Sigma_c \approx_{\mathcal{W}} A\backslash\Sigma_c$.* $\square$

The BSSNI *verification problem* (BSNNI-VP) is the following: given a GLTS A with $\Sigma = \Sigma_u \cup \Sigma_c$, decide whether A has the BSNNI property. The problem is known to be polynomial time [5] in the size of the ϵ-abstract automaton A^ϵ.

4 Control of Non-Interference

The previous non interference problems (SSNI-VP, BSNNI-VP) consist in *checking* whether a GLTS 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 behaviour of the high level user to ensure a given property. It is interesting because we can start with a very permissive high level user and then check whether its behaviour can be restricted by a controller to ensure a non-interference property.
The SSNI-*Control Problem* (SNNI-CP) is the following: given a GLTS A with $\Sigma_c \cup \Sigma_u$,

$$\textit{Is there a controller } C \textit{ s.t. } (C \times A)/\Sigma_c \approx_{\mathcal{L}} (C \times A)\backslash\Sigma_c\textit{?} \quad \text{(SNNI-CP)}$$

The SNNI-*Control Synthesis Problem* (SNNI-CSP) asks to compute a witness when the answer to the SNNI-CP is “yes”. The BSSNI-CP is defined in the obvious manner: given a GLTS A with $\Sigma_c \cup \Sigma_u$,

$$\textit{Is there a controller } C \textit{ s.t. } (C \times A)/\Sigma_c \approx_{\mathcal{W}} (C \times A)\backslash\Sigma_c\textit{?} \quad \text{(BSNNI-CP)}$$

Again the synthesis problem associated with the BSNNI-CP asks to compute a witness when the answer to the BSNNI-CP is “yes”. In the sequel we show how to solve Problems SNNI-CP and BSNNI-CP.

4.1 SNNI Control Problem

We reduce the SNNI-CP to the supervisory control problem. Let $\mathcal{U}$ be an automaton that accepts Σ_c^*. We first prove the following lemma:

Lemma 2.

$$(C \times A)/\Sigma_c \approx_{\mathcal{L}} (C \times A)\backslash\Sigma_c \iff \mathcal{L}(C \times A) \subseteq \mathcal{L}(A\backslash\Sigma_c \times \mathcal{U}) \cap \mathcal{L}(A).$$

Proof. By definition, $(C\times A)/\Sigma_c \approx_{\mathcal{L}} (C\times A)\backslash\Sigma_c \iff \mathcal{L}((C\times A)/\Sigma_c) = \mathcal{L}((C\times A)\backslash\Sigma_c)$. First notice that $\mathcal{L}((C\times A)\backslash\Sigma_c) = \mathcal{L}(A\backslash\Sigma_c)$. Indeed, any controller C for A cannot prevent uncontrollable actions from occurring. Moreover, a controller can only restrict the set of controllable actions from any state s and thus $(C\times A)\backslash\Sigma_c$ is just the LTS A where all the branches following a Σ_c action have been pruned. Notice also that $\mathcal{L}((C\times A)\backslash\Sigma_c) \subseteq \mathcal{L}((C\times A)/\Sigma_c)$. Using the previous two equations, we obtain:

$$(C \times A)/\Sigma_c \approx_{\mathcal{L}} (C \times A)\backslash\Sigma_c \iff \mathcal{L}((C \times A)/\Sigma_c) \subseteq \mathcal{L}(A\backslash\Sigma_c)$$

Moreover we can also prove that for any controller C for A:

$$(i)\, \mathcal{L}((C\times A)/\Sigma_c) \subseteq \mathcal{L}(A\backslash\Sigma_c) \iff (ii)\, \mathcal{L}(C\times A) \subseteq \mathcal{L}(A\backslash\Sigma_c\times\mathcal{U})\cap\mathcal{L}(A)$$

Assume (i) holds. Let $w \in \mathcal{L}(C\times A)$. Then $w/\Sigma_c \in \mathcal{L}(C\times A)/\Sigma_c = \mathcal{L}((C\times A)/\Sigma_c)$. By (i), $w/\Sigma_c \in \mathcal{L}(A\backslash\Sigma_c)$. Then w must be equal to w/Σ_c in which some Σ_c actions are inserted, which is exactly the definition of $\mathcal{L}(A\backslash\Sigma_c\times\mathcal{U})$. As $\mathcal{L}(C\times A) \subseteq \mathcal{L}(A)$, this entails $\mathcal{L}(C\times A) \subseteq \mathcal{L}(A\backslash\Sigma_c \times \mathcal{U})\cap\mathcal{L}(A)$. Assume (ii) holds. Let $w \in \mathcal{L}((C\times A)/\Sigma_c)$. By definition there is some $w' \in \mathcal{L}(C\times A)$ s.t. $w = w'/\Sigma_c$. By (ii) $w' \in \mathcal{L}(A\backslash\Sigma_c\times\mathcal{U})$. This entails $w \in \mathcal{L}(A\backslash\Sigma_c)$. □

This enables us to reduce the SNNI-CP to SCSLP:

Theorem 1. *The SNNI-CSP is in EXPTIME.*

Proof. The SNNI-CP asks whether there exists a controller C s.t. $(C\times A)\Sigma_c \approx_{\mathcal{L}} (C\times A)\backslash\Sigma_c$. Using Lemma 2, this problem amounts to finding a controller C s.t. $\mathcal{L}(C\times A) \subseteq \mathcal{L}(A\backslash\Sigma_c\times\mathcal{U})\cap\mathcal{L}(A)$. Thus we just need to solve an instance of the SCSLP with $K = \mathcal{L}(A\backslash\Sigma_c\times\mathcal{U})\cap\mathcal{L}(A)$. The right-hand side of this equation is a fixed (closed) language K that can be generated by a deterministic automaton A_K. Notice that as A may be non deterministic, this automaton has size at most exponential in the size of A. As stated in section 3.1, we can compute the most liberal controller s.t. C s.t. $(C\times A)/\Sigma_c \approx_{\mathcal{L}} (C\times A)\backslash\Sigma_c$ and moreover C is a finite memory controller of exponential size in the size of A. Indeed, C is a memoryless controller for a game $A \times \bar{A}_K$ where $\bar{A}_K$ generates the complement language of K. Notice that if A is deterministic the algorithm becomes polynomial because no determinization step is needed. $\square$

A consequence of Theorem 1 is that SNNI-VP can be solved by our algorithm: to solve SNNI-VP, we compute the most liberal controller C and then check that for each state (s,p) of $A \times \bar{A}_K$ we have $En(s,p) = En(s)$ *i.e.* the most liberal controller does not prevent any of the high level actions.

Example 2 (Control of SNNI). We use the automaton of Figure 2 (example 1) to show how to synthesize a controller that ensures SNNI. We recall that $\Sigma_c = \{h_1, h_2\}$ and $\Sigma_u = \{l_1, l_2\}$. First we have $\mathcal{L}(\mathcal{A}) = \{h_1.l_1, l_1, h_2.l_2\}$, $\mathcal{L}(\mathcal{A} \backslash \Sigma_c \times \mathcal{U}) = \Sigma_c^*.l_1.\Sigma_c^*$, and thus $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A} \backslash \Sigma_c \times \mathcal{U}) = \{h.l_1, l_1\}$.

We can use standard controller synthesis procedures[8] (see e.g. [14]) to compute the most liberal controller. This is a controller that avoids states $\{4,5\}$ in $\mathcal{A}$ and thus prevents h_2. The subsystem of $\mathcal{A}$ obtained by removing h_2 is the maximal subsystem that has the SNNI property and the most liberal controller C is given on Figure 3. $\blacksquare$

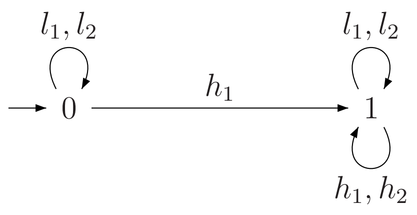
$$\begin{tikzcd}
\arrow[r] & 0 \arrow[loop above, "{l_1, l_2}"] \arrow[r, "h_1"] & 1 \arrow[loop above, "{l_1, l_2}"] \arrow[loop below, "{h_1, h_2}"]
\end{tikzcd}$$

Fig. 3. The Most Liberal Controller for $\mathcal{A}$

4.2 BSNNI Control Problem

We can define a satisfaction relation $\models_\varepsilon$ that considers the ε action as the invisible action and extend our interpretation of μ-calculus formulas over LTS containing ε by: $A \models_\epsilon \phi \iff A^\epsilon \models \phi$. If $L \subseteq \Sigma^\varepsilon$, we define

[8] This generates *state* based memoryless controllers which generates the supremal controllable sub-language of $\mathcal{A}$ w.r.t. $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A} \backslash \Sigma_c \times \mathcal{U})$.

the following "macro" operators for the μ-calculus:

$$\begin{array}{rr} [L]\varphi \overset{\text{def}}{=} \wedge_{b\in L}[b]\varphi & \langle L\rangle\varphi \overset{\text{def}}{=} \vee_{b\in L}\langle b\rangle\varphi \\ \langle L^*\rangle\varphi \overset{\text{def}}{=} \mu X.(\varphi \vee \langle L\rangle X) & [L^*]\varphi \overset{\text{def}}{=} \nu X.(\varphi \wedge [L]X) \end{array}$$

The translation $\kappa_L(\varphi)$ of a μ-formula φ w.r.t. $L \subseteq \Sigma^\varepsilon$ is inductively given by:

$$\begin{array}{ll} \kappa_L(Z) = Z & \kappa_L(\neg\varphi) = \neg\kappa_L(\varphi) \\ \kappa_L(\phi \wedge \psi) = \kappa_L(\phi) \wedge \kappa_L(\psi) & \kappa_L(\phi \vee \psi) = \kappa_L(\phi) \vee \kappa_L(\psi) \\ \kappa_L([\sigma]\varphi) = [L^*][\sigma][L^*]\kappa_L(\varphi) & \kappa_L(\langle\sigma\rangle\varphi) = \langle L^*\rangle\langle\sigma\rangle\langle L^*\rangle\kappa_L(\varphi) \\ \kappa_L(\nu Z.\phi) = \nu Z.\kappa_L(\varphi) & \kappa_L(\mu Z.\phi) = \mu Z.\kappa_L(\varphi) \end{array}$$

Lemma 3. *Let $L \subseteq \Sigma$. Then $A/L \models_\varepsilon \varphi \iff A \models \kappa_L(\varphi)$.*

Proof. It follows directly from Definitions (5) and (4) and $\models_\varepsilon$. □

Example 3. Let A to be the automaton of Figure 2 (example 1). We recall that $\Sigma_c = \{h_1, h_2\}$ and $\Sigma_u = \{l_1, l_2\}$. $CF(A\backslash\Sigma_c)$ is defined by the following equation system:

$$\begin{aligned} X_0 &= \langle l_1\rangle X_3 \wedge [l_1]X_3 \wedge [l_2]\mathsf{ff} \\ X_3 &= [l_1]\mathsf{ff} \wedge [l_2]\mathsf{ff} \end{aligned}$$

and $\kappa_{\Sigma_c}(CF(A))$ is defined by the following equation system:

$$\begin{aligned} X_0 &= \langle\Sigma_c^*\rangle\langle l_1\rangle\langle\Sigma_c^*\rangle X_3 \wedge [\Sigma_c^*][l_1][\Sigma_c^*]X_3 \wedge [\Sigma_c^*][l_2][\Sigma_c^*]\mathsf{ff} \\ X_3 &= [\Sigma_c^*][l_1][\Sigma_c^*]\mathsf{ff} \wedge [\Sigma_c^*][l_2][\Sigma_c^*]\mathsf{ff} \end{aligned}$$ ■

Moreover we can relate *weak bisimulation*, *hiding* and satisfiability of a characteristic formula:

Lemma 4. *Assume B is an automaton with no ε transitions and $L \subseteq \Sigma$. Then $A/L \approx_{\mathcal{W}} B \iff A \models \kappa_L(CF(B))$.*

Proof.

$$\begin{aligned} A/L \approx_{\mathcal{W}} B &\iff (A/L)^\varepsilon \approx_{\mathcal{S}} B \quad \text{[by definition of } \approx_{\mathcal{W}}] \\ &\iff (A/L)^\varepsilon \models CF(B) \quad \text{[by Lemma 1]} \\ &\iff A/L \models_\varepsilon CF(B) \quad \text{[by definition of } \models_\varepsilon] \\ &\iff A \models \kappa_L(CF(B)) \quad \text{[by Lemma 3]} \end{aligned}$$ □

Using the previous lemmas and the characteristic formula $CF(A)$ of a LTS, we can reduce the BSNNI-CP to a μ-CP and state the following theorem:

Theorem 2. *BSNNI-CP is decidable and if the answer to the BSNNI-CP is "yes" a most permissive controller can be effecively synthesized* i.e. *BSNNI-CSP is computable.*

Proof. The BSNNI-CP is the following:

$$\textit{Is there any controller } C \textit{ for } A \textit{ s.t. } (C \times A)/\Sigma_c \approx_{\mathcal{W}} (C \times A)\backslash\Sigma_c\textit{?} \quad (2)$$

As C does not restrict the uncontrollable actions, we have again that $(C \times A)\backslash\Sigma_c = A\backslash\Sigma_c$. We can then compute the characteristic formula for $A\backslash\Sigma_c$ and obtain $CF(A\backslash\Sigma_c)$. Applying Lemma 4 we obtain:

$$\begin{aligned}(C \times A)/\Sigma_c \approx_{\mathcal{W}} (C \times A)\backslash\Sigma_c &\iff (C \times A)/\Sigma_c \approx_{\mathcal{W}} A\backslash\Sigma_c \\ &\iff (C \times A) \models \kappa_{\Sigma_c}(CF(A\backslash\Sigma_c))\end{aligned}$$

This enables us to reduce the BSSNI-CP to the following μ-CP:

$$\exists C \text{ s.t. } C \times A \models \kappa_{\Sigma_c}(CF(A\backslash\Sigma_c))$$

We have now a μ-CP to solve of the form $\exists C$ s.t. $C \times A \models \varphi$ with φ a μ-calculus formula. This can be done as stated in [4, 13]. If A is controllable w.r.t. $\kappa_{\Sigma_c}(CF(A\backslash\Sigma_c))$ we can build a finite memory most permissive controller that satisfies BSNNI. □

As stated in section 3 the complexity of μ-CP is not yet known and thus we cannot obtain complexity result with our reduction.

Example 4 (Control of BSNNI). We use the automaton of Figure 2 (example 1) to show how to synthesize a controller that ensures BSNNI. There are four possibilities for the most permissive controller C: either it allows $\{h_1, h_2\}$ or $\{h_1\}$ or $\{h_2\}$ or nothing. If it allows h_2, the initial state of $A \times C$ does not satisfy $[\Sigma_c^*][l_2][\Sigma_c^*]\mathsf{ff}$ in X_0 because after h_2 an action l_2 is enabled which is forbidden by $\kappa_{\Sigma_c}(CF(A\backslash\Sigma_c))$. If it allows l_1 $\mathcal{A} \times C$ satisfies $\kappa_{\Sigma_c}(CF(A\backslash\Sigma_c))$ and thus the most permissive controller is the one given by Figure 3 again. ■

5 Conclusion

In this paper we have defined the control problems SNNI-CP and BSNNI-CP for the two types of non-interference properties SNNI and BSNNI. We have proved that SNNI-CP and BSNNI-CP are decidable and we solved both associated control synthesis problems *i.e.* given a system S, compute the *most permissive* non-interferent (sub)system of S.

Our future work will consist in:

– find the exact complexity of both SNNI-CP and BSNNI-CP;
– implement our framework. For SNNI there is BDD based tool [7] to solve the supervisory control problem. Concerning BSNNI and the μ-CP, a tool Synthesis implementing the theoretical setting of [4] has been implemented [10];
– extend our results to timed systems and timed non interference to generalize and refine the results of [6].

References

1. L. Aceto and A. Ingólfsdóttir. Characteristic formulæ: From automata to logic. Technical Report RS-07-2, BRICS, Aarhus, Denmark, Jan. 2007.
2. J. Agat. Transforming out timing leaks. In *POPL*, pages 40–53, 2000.
3. A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.
4. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 1(303):7–34, 2003.
5. R. Focardi and R. Gorrieri. Classification of security properties (part I: Information flow). In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design I: FOSAD 2000 Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*, pages 331–396, Heidelberg, 2001. Springer-Verlag.
6. G. Gardey, J. Mullins, and O. H. Roux. 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, Aug. 2005. Elsevier.
7. G. Hoffmann and H. Wong-Toi. Symbolic synthesis of supervisory controllers. In *Proc. of the American Control Conference, Chicago, IL, June 1992*, pages 2789–2793, 1992.
8. D. Kozen. Results on the propositional -calculus. In M. Nielsen and E. M. Schmidt, editors, *ICALP'82*, volume 140, pages 348–359, Aarhus, Denmark, 1982. Springer.
9. M. Müller-Olm. Derivation of characteristic formulæ. In *Workshop on Mathemantical Foundations of Computer Science*, volume 18 of *Electronic Notes in Theoretical Computer Science*, Brno, Slovakia, 1998. Elsevier, Amsterdam.
10. G. Point. The synthesis toolbox - from modal automata to controller synthesis. Technical Report 1342-05, LaBRI, 2005. available at http://www.labri.fr/publications/mvtsi/2005/Poi05.
11. P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1), Jan. 1987.

12. P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, Jan. 1989.
13. S. Riedweg and S. Pinchinat. Quantified mu-calculus for control synthesis. In *28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03), Bratislava, Slovakia, August 25-29, 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 642–651, 2003.
14. W. Thomas. On the synthesis of strategies in infinite games. In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, pages 1–13. Springer, 1995. Invited talk.
15. R. van der Meydena and C. Zhang. 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*, pages 61–75. Elsevier, 2006.