Synthesis of Non-Interferent Timed Systems

Gilles Benattar, Franck Cassez, Didier Lime, Olivier Henri Roux

Synthesis of Non-Interferent Timed Systems*

Gilles Benattar[1], Franck Cassez[2,**], Didier Lime[1], Olivier H.Roux[1]

[1] IRCCyN/CNRS, BP 92101, 1 rue de la Noë, 44321 Nantes Cedex 3, France
[2] National ICT Australia & CNRS, Univ. of New South Wales, Sydney, Australia

Abstract. In this paper, we focus on the synthesis of secure timed systems which are given by timed automata. The security property that the system must satisfy is a *non-interference* property. Various notions of non-interference have been defined in the literature, and in this paper we focus on *Strong Non-deterministic Non-Interference* (SNNI) and we study the two following problems: (1) check whether it is possible to enforce a system to be SNNI; if yes (2) compute a subsystem which is SNNI.

Key words: Non-Interference, Synthesis, Timed Automaton.

1 Introduction

Modern computing environments allow the use of programs that are sent or fetched from different sites. Such programs may deal with secret information such as private data (of a user) or as classified data (of an organization). One of the basic concerns in such context is to ensure that the programs do not leak sensitive data to a third party, either maliciously or inadvertently. This 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 capture any causal dependency between high-level and low-level behaviors.

We assume that there are two users and the set of actions of the system S is partitioned into Σ_h (high-level actions) and Σ_l (low-level actions). The non-interference properties, namely strong non-deterministic non-interference (SNNI), cosimulation-based strong non-deterministic non-interference (CSNNI) and bisimulation-based strong non-deterministic non-interference (BSNNI), are out of the scope of the common safety/liveness classification of system properties [1]. The *non-interference verification problem*, for a given system S, consists in checking whether S is non-interferent.

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. For example, the Needham-Schroeder protocol can be proved unsecure by defining the security property using SNNI [3].

* Work supported by the French Government undergrant ANR-SETI-003.
** Author supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme.

In this paper, we consider the problem of *synthesizing* non-interferent timed systems. In contrast to verification, the *non-interference synthesis problem* assumes the system is *open i.e.*, we can restrict the behaviors of S: some events, in a particular set Σ_c, of S can be disabled. The *non-interference control problem* for a system S asks the following: "Is there a controller C s.t. $C(S)$ is non-interferent?" The associated *non-interference controller synthesis problem* asks to compute a witness mapping C.

Related Work. In [4] the authors consider the complexity of many non-interference *verification* problems but synthesis is not addressed. In [5] an exponential time decision procedure for checking whether a finite state system satisfies a given Basic Security Predicate (BSP) is presented but the synthesis problem is not addressed. There is also a large body of work on the use of static analysis techniques to enforce information flow policies. A general overview can be found in [6]. The non-interference synthesis problem was first considered in [7] for dense-time systems specified by timed automata. The non-interference property considered in [7] is the *state* non-interference property, which is less demanding than the one we consider here.

This paper is a follow-up of our previous work [8] about *non-interference control problems* for untimed systems. In [8], we assumed that the security domains coincided with the controllable and uncontrollable actions: high-level actions (Σ_h) could be disabled ($\Sigma_c = \Sigma_h$) whereas low-level actions (Σ_l) could not. We studied the synthesis problems for SNNI and BSNNI and proved they are decidable. In the present paper we extend the previous work in two directions: (1) we release the constraint $\Sigma_c = \Sigma_h$ and (2) consider the synthesis problem for timed automata. Nevertheless we restrict the class of non-interference properties to SNNI.

The motivations for this work are manyfold. Releasing $\Sigma_c = \Sigma_h$ 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_l can be disabled. We can view actions of Σ_l as capabilities of the low-level user (*e.g.*, pressing a button), and it thus makes sense to prevent the user from using the button for instance by disabling/hiding it temporarily.

It is also of theoretical interest, because this non-interference synthesis problem is *really* more difficult than the corresponding verification problem in the sense that we can *reduce* the SNNI verification problem to a particular instance of the synthesis problem: we just have to take $\Sigma_c = \emptyset$. This was not the case for the versions of the synthesis problems studied in [8].

We start by studying the SNNI synthesis problem for timed automata because SNNI is a rather simple notion of interference. Still as said earlier, it is expressive enough for example to prove that the Needham-Schroeder protocol is flawed [3]. Controller synthesis enables one to find automatically the patch(es) to apply to make such a protocol secure. The use of dense-time to model the system clearly gives a more accurate and realistic model for the system and a potential attacker that can measure time.

Our Contribution. In this paper, we first exhibit a class *dTA* of timed automata for which the SNNI verification problem is decidable. The other results are: (1) we prove that deciding whether there is controller C for A such that (s.t. in the following) $C(A)$ is SNNI, is decidable for the previous class *dTA*; (2) we reduce the SNNI synthesis problem to solving a sequence of *safety timed games*.

Organization of the paper. Section 2 recalls the basics of timed automata, timed languages and some results on safety timed games. Section 3 gives the definition of the non-interference synthesis problem we are interested in. Section 4 contains the main result: we show that there is a largest subsystem which is SNNI and this subsystem is effectively computable. Finally, in Section 5 we conclude and give a list of open problems and future work.

2 Preliminaries

Let $\mathbb{R}_+$ be the set of non-negative reals and $\mathbb{N}$ the set of integers. Let X be a finite set of positive real-valued variables called *clocks*. A valuation of the variables in X is a function $X \to \mathbb{R}_+$, that can be written as a vector of $\mathbb{R}_+^X$. We let $\mathbf{0}_X$ be the valuation s.t. $\mathbf{0}_X(x) = 0$ for each $x \in X$ and use $\mathbf{0}$ when X is clear from the context. Given a valuation v and $R \subseteq X$, $v[R \mapsto 0]$ is the valuation s.t. $v[R \mapsto 0](x) = v(x)$ if $x \notin R$ and 0 otherwise. An atomic constraint (over X) is of the form $x \sim c$, with $x \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. A (convex) formula is a conjunction of atomic constraints. $\mathcal{C}(X)$ is the set of convex formulas. Given a valuation v (over X) and a formula γ over X, $\gamma(v)$ is the truth value, in $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$, of γ when each symbol x in γ is replaced by $v(x)$. If $t \in \mathbb{R}_+$, we let $v + t$ be the valuation s.t. $(v + t)(x) = v(x) + t$. Let $|V|$ be the cardinality of the set V.

Let Σ be a finite set, $\varepsilon \notin \Sigma$ and $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$. A *timed word* w over Σ is a sequence $w = (\delta_0, a_0)(\delta_1, a_1) \cdots (\delta_n, a_n)$ s.t. $(\delta_i, a_i) \in \mathbb{R}_+ \times \Sigma$ for $0 \leq i \leq n$ where δ_i represents the amount of time elapsed[3] between a_{i-1} and a_i. $T\Sigma^*$ is the set of timed words over Σ. We denote by uv the *concatenation* of two timed words u and v. As usual ε is also the empty word s.t. $(\delta_1, \varepsilon)(\delta_2, a) = (\delta_1 + \delta_2, a)$. Given a timed word $w \in T\Sigma^*$ and $L \subseteq \Sigma$ the *projection* of w over L is denoted by $\pi_L(w)$ and is defined by $\pi_L(w) = (\delta_0, b_0)(\delta_1, b_1) \cdots (\delta_n, b_n)$ with $b_i = a_i$ if $a_i \in L$ and $b_i = \varepsilon$ otherwise. The *untimed* projection of w, *Untimed*(w), is the word $a_0 a_1 \cdots a_n$ of Σ^*.

A *timed language* is a subset of $T\Sigma^*$. Let L be a timed language, the untimed language of L is *Untimed*$(L) = \{v \in \Sigma^* \mid \exists w \in L \; s.t. \; v = \textit{Untimed}(w)\}$.

A *timed transition system (TTS)* is a tuple $\mathcal{S} = (S, s_0, \Sigma^\varepsilon, \to)$ where S is a set of states, s_0 is the initial state, Σ a finite alphabet of actions, $\to \subseteq S \times \Sigma^\varepsilon \cup \mathbb{R}_+ \times S$ is the transition relation. We use the notation $s \xrightarrow{e} s'$ if $(s, e, s') \in \to$ and impose that for each $s \in S, s \xrightarrow{0} s$.

A run ρ of $\mathcal{S}$ from s is a finite sequence of transitions $\rho = q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} q_n$ s.t. $q_0 = s_0$ and $(q_i, e_i, q_{i+1}) \in \to$ for $0 \leq i \leq n-1$. We denote by *last*(ρ) the last state of the sequence *i.e.*, the state q_n. We let *Runs*$(s, \mathcal{S})$ be the set of runs from s in $\mathcal{S}$ and *Runs*$(\mathcal{S})$ = *Runs*$(s_0, \mathcal{S})$. We write $q_0 \xrightarrow{*} q_n$ if there is a run from q_0 to q_n. The set of *reachable* states in *Runs*$(\mathcal{S})$ is *Reach*$(\mathcal{S}) = \{s \mid s_0 \xrightarrow{*} s\}$. Each run can be written in a normal form where delay and discrete transitions alternate *i.e.*, $\rho = q_0 \xrightarrow{\delta_0} \xrightarrow{e_0} q_1 \xrightarrow{\delta_1} \xrightarrow{e_1} \cdots \xrightarrow{\delta_n} \xrightarrow{e_n} q_{n+1} \xrightarrow{\delta} q'_{n+1}$. The *trace* of ρ is *trace*$(\rho) = (\delta_0, e_0)(\delta_1, e_1) \cdots (\delta_n, e_n)$.

[3] For $i = 0$ this is the amount of time since the system started.

A *timed automaton (TA)* is a tuple $A = (Q, q_0, X, \Sigma^\varepsilon, E, Inv)$ where: $q_0 \in Q$ is the initial location; X is a finite set of positive real-valued clocks; Σ^ε is a finite set of actions; $E \subseteq Q \times \mathcal{C}(X) \times \Sigma^\varepsilon \times 2^X \times Q$ is a finite set of edges. An edge (q, γ, a, R, q') goes from q to q', with the guard $\gamma \in \mathcal{C}(X)$, the action a and the reset set $R \subseteq X$; $Inv : Q \to \mathcal{C}(X)$ is a function that assigns an invariant to any location; we require that the atomic formulas of an invariant are of the form $x \sim c$ with $\sim \in \{<, \leq\}$.

A finite (or untimed) automaton $A = (Q, q_0, \Sigma^\varepsilon, E)$ is a special kind of timed automaton with $X = \emptyset$, and consequently all the guards and invariants are vacuously true. A timed automaton A is *deterministic* if for $(q_1, \gamma, a, R, q_2), (q_1, \gamma', a, R', q_2') \in E, \gamma \wedge \gamma' \neq \mathsf{false} \Rightarrow q_2 = q_2'$ *and* $R = R'$. We recall that timed automata cannot always be determinized (*i.e.*, find a deterministic TA which accepts the same language as a non-deterministic one, see [9]), and moreover, checking whether a timed automaton is determinizable is undecidable [10]. The *semantics* of a timed automaton $A = (Q, q_0, X, \Sigma^\varepsilon, E, Inv)$ is the TTS $\mathcal{S}^A = (Q \times \mathbb{R}_+^X, (q_0, \mathbf{0}), \Sigma^\varepsilon, \to)$ defined in the usual way. If $s = (q, v)$ is a state of $\mathcal{S}^A$, we denote by $s + \delta$ the (only) state reached after δ time units, *i.e.*, $s + \delta = (q, v + \delta)$. The sets of runs of A is defined as *Runs*(A) = *Runs*$(\mathcal{S}^A)$ where $\mathcal{S}^A$ is the semantics of A. A timed word $w \in T\Sigma^*$ is *generated* by A if $w =$ *trace*(ρ) for some $\rho \in$ *Runs*(A). The timed language generated by A, $\mathcal{L}(A)$, is the set of timed words generated by A. Two automata A and B are *language equivalent*, denoted by $A \approx B$, if $\mathcal{L}(A) = \mathcal{L}(B)$ *i.e.*, they generate the same set of timed words.

Let $A_1 = (Q_1, q_{01}, X_1, \Sigma^\varepsilon, E_1, Inv_1)$ and $A_2 = (Q_2, q_{02}, X_2, \Sigma^\varepsilon, E_2, Inv_2)$ be two TA with $X_1 \cap X_2 = \emptyset$. Let $L \subseteq \Sigma$. The *synchronized product* of A_1 and A_2 w.r.t. L, is the timed automaton $A_1 \times_L A_2 = (Q, q_0, X, \Sigma^\varepsilon, E, Inv)$ defined in the usual way (synchronization occurs only for actions in L). When it is clear from the context we omit the subscript L in $\times_L$.

In the sequel we will use two operators on TA: the first one gives an *abstracted* automaton and simply hides a set of labels $L \subseteq \Sigma$. Given a TA $A = (Q, q_0, X, \Sigma^\varepsilon, E, Inv)$ and $L \subseteq \Sigma$ we define the TA $A/L = (Q, q_0, X, (\Sigma \backslash L)^\varepsilon, E_L, Inv)$ where $(q, \gamma, a, R, q') \in E_L \iff (q, \gamma, a, R, q') \in E$ for $a \in \Sigma \backslash L$ and $(q, \gamma, \varepsilon, R, q') \in E_L \iff (q, \gamma, a, R, q') \in E$ for $a \in L \cup \{\varepsilon\}$. The *restricted* automaton cuts transitions labeled by the letters in $L \subseteq \Sigma$: Given a TA $A = (Q, q_0, X, \Sigma, E, Inv)$ and $L \subseteq \Sigma$ we define the TA $A \backslash L = (Q, q_0, X, \Sigma \backslash L, E_L, Inv)$ where $(q, \gamma, a, R, q') \in E_L \iff (q, \gamma, a, R, q') \in E$ for $a \in \Sigma \backslash L$.

We will also use some results on safety control for timed games which have been introduced and solved in [11].

A *Timed Game Automaton (TGA)* $A = (Q, q_0, X, \Sigma, E, Inv)$ is a timed automaton with its set of actions Σ partitioned into *controllable* (Σ_c) and *uncontrollable* (Σ_u) actions. Let A be a TGA and *Bad* $\subseteq Q \times \mathbb{R}_+^X$ be the set of bad states to avoid. *Bad* can be written $\cup_{1 \leq i \leq k} (\ell_i, Z_i)$, with each Z_i defined as a conjunction of formulas of $\mathcal{C}(X)$ and each $\ell_i \in Q$. The *safety control problem* for $(A,$ *Bad*$)$ is: decide wether there is a controller to constantly avoid *Bad*. Let λ be a fresh special symbol not in Σ^ε denoting the action "do nothing".

A *controller* C for A is a partial function from *Runs*(A) to $2^{\Sigma_c \cup \{\lambda\}}$. We require that $\forall \rho \in$ *Runs*(A), if $a \in C(\rho) \cap \Sigma_c$ then *last*$(\rho) \xrightarrow{a} (q', v')$ for some (q', v') and

if $\lambda \in C(\rho)$ then $\mathit{last}(\rho) \xrightarrow{\delta} (q', v')$ for some $\delta > 0$. A controller C is *state-based* or *memoryless* whenever $\forall \rho, \rho' \in \mathit{Runs}(A), \mathit{last}(\rho) = \mathit{last}(\rho')$ implies that $C(\rho) = C(\rho')$.

Remark 1. We assume a controller gives a set of actions that are enabled which differs from standard definitions [11] where a controller only gives one action. Nevertheless for safety timed games, one computes a most permissive controller (if there is one) which gives for each state the largest set of actions which are safe. It follows that any reasonable (*e.g.*, Non-Zeno) sub-controller of this most permissive controller avoids the set of bad states.

$C(A)$ defines "A supervised/restricted by C" and is inductively defined by its set of runs:

- $(q_0, \mathbf{0}) \in \mathit{Runs}(C(A))$,
- if $\rho \in \mathit{Runs}(C(A))$ and $\rho \xrightarrow{e} s' \in \mathit{Runs}(A)$, then $\rho \xrightarrow{e} s' \in \mathit{Runs}(C(A))$ if one of the following three conditions holds:
  1. $e \in \Sigma_u$,
  2. $e \in \Sigma_c \cap C(\rho)$,
  3. $e \in \mathbb{R}_+$ and $\forall \delta \; s.t. \;\; 0 \leq \delta < e, \mathit{last}(\rho) \xrightarrow{\delta} \mathit{last}(\rho) + \delta \wedge \lambda \in C(\rho \xrightarrow{\delta} \mathit{last}(\rho) + \delta)$.

$C(A)$ can also be viewed as a TTS where each state is a run of A and the transitions are given by the previous definition. C is a winning controller for (A, Bad) if $\mathit{Reach}(C(A)) \cap \mathit{Bad} = \emptyset$. For safety timed games, the results are the following [11,12]:

- it is decidable (EXPTIME-complete) whether there is a winning controller for a safety game (A, Bad);
- in case there is one, there is a *most permissive* controller which is memoryless on the region graph of the TGA A. This most permissive controller can be represented by a TA. This also means that the set of runs of $C(A)$ is itself the semantics of a timed automaton, that can be effectively built from A.

3 Non-Interference Synthesis Problem

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. In the sequel, we assume $A = (Q, q, X, \Sigma_h \cup \Sigma_l, E, Inv)$ is a timed automaton where Σ_l (resp. Σ_h) is the set of *public* (resp. *private*) actions and we let $\Sigma = \Sigma_h \cup \Sigma_l$.

Definition 1 (SNNI). *A has the* strong non-deterministic non-interference *property (in short "A is SNNI") if $A/\Sigma_h \approx A \backslash \Sigma_h$.* □

The SNNI verification problem (SNNI-VP) asks to check whether a system A is SNNI.

Example 1 (SNNI). Figure 1 gives examples of systems $A(k)$ which are SNNI and not SNNI depending on the value of integer k. The high-level actions are $\Sigma_h = \{h\}$ and the low-level actions are $\Sigma_l = \{l\}$. (δ, l) with $1 \leq \delta < 2$ is a trace of $A(1)/\Sigma_h$ but not of $A(1)\backslash\Sigma_h$ and so, $A(1)$ is not SNNI. $A(2)$ is SNNI as we can see that $A(2)/\Sigma_h \approx A(2)\backslash\Sigma_h$. Note that $A(k)$ without the clock constraints, then it is SNNI. ■

Remark 2. Let $\mathcal{L}_h = \Sigma_l^* \Sigma_h \Sigma^*$. Then $\mathcal{L}(A\backslash\Sigma_h) = \mathcal{L}(A)\backslash\mathcal{L}_h$. Also $\mathcal{L}(A/\Sigma_h) = \pi_{\Sigma_l}(\mathcal{L}(A))$. This shows that SNNI is really a language property as if $\mathcal{L}(A) = \mathcal{L}(B)$, then A is SNNI iff B is SNNI.

We anticipate on the definition of the SNNI control problem (SNNI-CP): in general, control problems are more difficult than the corresponding verification problems and this is the case for the SNNI-CP. Thus we cannot expect to solve the SNNI-CP if the SNNI-VP is undecidable. The SNNI-VP for TA consists in checking whether a TA is SNNI. It was proved in [7] that the SNNI-VP is undecidable for non deterministic timed automata. We first refine this result and exhibit a class of TA for the SNNI-VP is decidable. Let *dTA* be the set of TA A s.t. $A\backslash\Sigma_h$ is deterministic (membership in *dTA* can be checked syntactically).

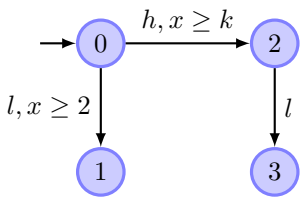

Fig. 1. Automaton $A(k)$

Theorem 1. *The SNNI-VP is PSPACE-complete for dTA.*

The proof of PSPACE-hardness consists in reducing the language inclusion problem $\mathcal{L}(B) \subseteq \mathcal{L}(A)$ with A a deterministic TA, which is PSPACE-complete [9], to the SNNI-VP. PSPACE-easiness is proved along the same lines.

Proof. $\mathcal{L}(A\backslash\Sigma_h) \subseteq \mathcal{L}(A/\Sigma_h)$ is always true. Checking whether $\mathcal{L}(B) \subseteq \mathcal{L}(A)$ with A a deterministic TA is PSPACE-complete [9]. Thus checking if $\mathcal{L}(A/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$ can be done in PSPACE for *dTA* and the SNNI-VP is thus in PSPACE.
For PSPACE-hardness, we reduce the language inclusion problem $\mathcal{L}(B) \subseteq \mathcal{L}(A)$ with A a deterministic TA, to the SNNI-VP. Let A be a deterministic timed automaton (initial location i_A) defined over Σ_l and B a timed automaton (initial location i_B) defined over Σ_l. We let D be the timed automaton over $\Sigma_l \cup \Sigma_h$, with $\Sigma_h = \{h\}$ and $h \notin \Sigma_l$, defined as in Figure 2. It follows that $\mathcal{L}(B) \subseteq \mathcal{L}(A) \iff$ *D is SNNI.* $\Longrightarrow$ holds because if $\mathcal{L}(B) \subseteq \mathcal{L}(A)$ then $\mathcal{L}(D/\Sigma_h) = \mathcal{L}(B) \cup \mathcal{L}(A) = \mathcal{L}(A) = \mathcal{L}(D\backslash\Sigma_h)$. For the converse $\Longleftarrow$, if D is SNNI $\mathcal{L}(A) \cup \mathcal{L}(B) = \mathcal{L}(D/\Sigma_h)$, and as $\mathcal{L}(D/\Sigma_h) = \mathcal{L}(D\backslash\Sigma_h) = \mathcal{L}(A)$ we get $\mathcal{L}(B) \subseteq \mathcal{L}(A)$. □

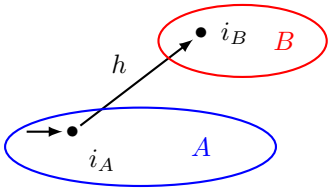

Fig. 2. Timed Automaton D

This reduction of language inclusion to the SNNI-VP also holds for finite automata. For finite non-deterministic automata, checking language inclusion is in PSPACE [13]. It follows that the SNNI-VP is in PSPACE for non-deterministic $A\backslash\Sigma_h$ and in PTIME for deterministic $A\backslash\Sigma_h$. Table 1 summarizes the results for the SNNI-VP.

The previous non-interference verification problem (SNNI-VP), consists in *checking* whether an automaton A has the non-interference property. If the answer is "no", one has to investigate why the non-interference property is not true, modify A and check the property again. In contrast to the verification problem, the synthesis problem indicates whether there is a way of restricting the behavior of users to ensure a given

	Finite Automata	Timed Automata
$A\backslash\Sigma_h$ Deterministic	PTIME	PSPACE-Complete
$A\backslash\Sigma_h$ Non-Deterministic	PSPACE-Complete	Undecidable [7]

Table 1. Results for the SNNI-VP

property. Thus we consider that only some actions in the set Σ_c, with $\Sigma_c \subseteq \Sigma_h \cup \Sigma_l$, are controllable and can be disabled. We let $\Sigma_u = \Sigma \setminus \Sigma_c$ denote the actions that are uncontrollable and thus cannot be disabled.

Recall that a *controller* C for A gives for each run ρ of A the set $C(\rho) \in 2^{\Sigma_c \cup \{\lambda\}}$ of actions that are enabled after this particular run. The SNNI-*Control Problem* (SNNI-CP) we are interested in is the following:

Is there a controller C s.t. $C(A)$ is SNNI ? (SNNI-CP)

The SNNI-*Controller Synthesis Problem* (SNNI-CSP) asks to compute a witness when the answer to the SNNI-CP is "yes".

4 Algorithms for the Synthesis Problems

4.1 Preliminary Remarks

First we motivate our definition of controllers which are mappings from *Runs*(A) to $2^{\Sigma_c \cup \{\lambda\}}$. The common definition of a controller in the literature is a mapping from *Runs*(A) to $\Sigma_c \cup \{\lambda\}$. Indeed, for the safety (or reachability) control problem, one can compute a mapping M : *Runs*$(A) \to 2^{\Sigma_c \cup \{\lambda\}}$ (most permissive controller), and a controller C ensures the safety goal iff $C(\rho) \in M(\rho)$. This implies that any sub-controller of M is a good controller. This is not the case for SNNI, even for finite automata, as the following example shows.

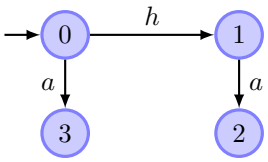

Fig. 3. Automaton D

Example 2. Let us consider the automaton D of Figure 3 with $\Sigma_c = \{a, h\}$. The largest sub-system of D which is SNNI is D itself. Disabling a from state 0 will result in an automaton which is not SNNI. ■

We are thus interested in computing the largest (if there is such) sub-system of A that we can control which is SNNI. Second, in our definition we allow a controller to forbid any controllable action. In contrast, in the literature, a controller should ensure some liveness and never block the system. In the context of security property, it makes sense to disable everything if the security policy cannot be enforced otherwise. This makes the SNNI-CP easy for finite automata.

Theorem 2. *For finite automata, the SNNI-CP is PSPACE-Complete.*

The proof consists in proving that if a finite automaton can be restricted to be SNNI, then disabling all the Σ_c actions is a solution. Thus the SNNI-CP reduces to the SNNI-VP and the result follows.

Proof. As time is not taken into acount in untimed automaton, we can have $C(\rho) = \emptyset$ for finite automaton (for general timed automaton, this would mean that we block the time.) The proof of the theorem consists in proving that if a finite automaton can be restricted to be SNNI, then disabling all the Σ_c actions is a solution. Let $C_\forall$ be the controller defined by $C_\forall(\rho) = \emptyset$. We prove the following: if C is a controller s.t. $C(A)$ is SNNI, then $C_\forall(A)$ is SNNI.

Assume a finite automaton D is SNNI. Let $e \in \Sigma_h \cup \Sigma_l$ and let $\mathcal{L}_e$ be the set of words containing at least one e. Depending on the type of e we have:

- if $e \in \Sigma_l$, then $\mathcal{L}((D\backslash\{e\})\backslash\Sigma_h) = \mathcal{L}(D\backslash\Sigma_h) \setminus \mathcal{L}_e$ and as D is SNNI, it is also equal to $\mathcal{L}(D/\Sigma_h) \setminus \mathcal{L}_e = \mathcal{L}((D\backslash\{e\})/\Sigma_h)$;
- if $e \in \Sigma_h$, $\mathcal{L}((D\backslash\{e\})/\Sigma_h) \subseteq \mathcal{L}(D/\Sigma_h) = \mathcal{L}(D\backslash\Sigma_h) = \mathcal{L}((D\backslash\{e\})\backslash\Sigma_h)$.

So, if D is SNNI, $D\backslash L$ is SNNI, $\forall L \subseteq \Sigma$. By remark 2, since $\mathcal{L}(C_\forall(D)) = \mathcal{L}(D\backslash\Sigma_c)$, if D is SNNI, then $D\backslash\Sigma_c$ is also SNNI and therefore $C_\forall(D)$ is SNNI.

Let A be the TA we want to restrict. Assume there is a controller C s.t. $C(A)$ is SNNI. $C_\forall(C(A))$ is SNNI so $C_\forall(C(A)) = C_\forall(A)$ is also SNNI which means that $A\backslash\Sigma_c$ is SNNI. This proves that: $\exists C$ s.t. $C(A)$ is SNNI $\Leftrightarrow$ $A\backslash\Sigma_c$ is SNNI.

It is then equivalent to check that $A\backslash\Sigma_c$ is SNNI to solve the SNNI-CP for A and this can be done in PSPACE. PSPACE-hardness comes from the reduction of SNNI-VP to SNNI-CP, by taking $\Sigma_c = \emptyset$. □

Theorem 2 does not hold for timed automata as the following example demonstrates.

Example 3. Figure 4 gives an example of a timed automaton H with high-level actions $\Sigma_h = \{h\}$ and low-level actions $\Sigma_l = \{a, b\}$. Assume $\Sigma_c = \{a\}$. Notice that $H\backslash\Sigma_c$ is not SNNI. Let the state based controller C be defined by: $C(0, x) = \{a, \lambda\}$ when H is in state $(0, x)$ with $x < 4$; and $C(0, x) = \{a\}$ when $x = 4$. Then $C(H)$ is SNNI. In this example, when $x = 4$ we prevent time from elapsing by forcing the firing of a which indirectly disables action h. To do this we just have to add an invariant $[x \leq 4]$ to location 0 of H and this cuts out the dashed transitions rendering $C(H)$ SNNI. ■

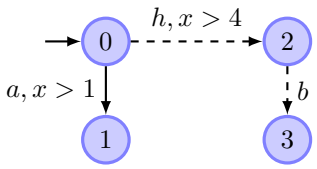

Fig. 4. The Automaton H

4.2 Algorithms for SNNI-CP and SNNI-CSP

In this section we first prove that the SNNI-CP is EXPTIME-hard for *dTA*. Then we give an EXPTIME algorithm to solve the SNNI-CP and SNNI-CSP.

Theorem 3. *For dTA, the SNNI-CP is EXPTIME-Hard.*

Proof. The safety control problem for TA is EXPTIME-hard [14]. In the proof of this theorem, T.A. Henzinger and P.W. Kopke use timed automata where the controller chooses an action and the environment resolves non-determinism. The hardness proof reduces the halting problem for alternating Turing Machines using polynomial space to a safety control problem. In our framework, we use TA with controllable and uncontrollable actions. It is not difficult to adapt the hardness proof of [14] to TA which are deterministic w.r.t. Σ_c actions and non deterministic w.r.t. Σ_u actions. As Σ_u transitions can never be disabled (they act only as spoiling actions), we can use a different label for each uncontrollable transition without altering the result in our definition of the safety control problem. Hence: the safety control problem as defined in section 2 is EXPTIME-hard for deterministic TA (with controllable and uncontrollable transitions). This problem can be reduced to the safety control problem of TA with only one state bad. We can now reduce the safety control problem for deterministic TA which is EXPTIME-hard to the SNNI control problem on *dTA*. Let $A = (Q \cup \{bad\}, q_0, X, \Sigma_c \cup \Sigma_u, E, Inv)$ be a TGA, with Σ_c (resp. Σ_u) the set of controllable (resp. uncontrollable) actions, and bad a location to avoid. We define A' by adding to A two uncontrollable transitions: $(bad, \mathsf{true}, h, \emptyset, q_h)$ and $(q_h, \mathsf{true}, l, \emptyset, q_l)$ where q_h and q_l are fresh locations with invariant true. l and h are two fresh uncontrollable actions in A'. We now define $\Sigma_h = \{h\}$ and $\Sigma_l = \Sigma_c \cup \Sigma_u \cup \{l\}$ for A'. By definition of A', for any controller C, if location *Bad* is not reachable in $C(A')$, then the actions h and then l can not be fired. Thus if there is controller for C for A which avoids *Bad*, the same controller C renders A' SNNI. Now if there is a controller C' s.t. $C'(A')$ is SNNI, it must never enable h: otherwise a (untimed) word $w.h.l$ would be in *Untimed*$(\mathcal{L}(C'(A')/\Sigma_h))$ but as no untimed word containing an l can be in *Untimed*$(\mathcal{L}(C'(A')\backslash\Sigma_h))$, and thus $C'(A')$ would not be SNNI. Notice that it does not matter whether we require the controllers to be non blocking (mappings from *Runs*(A) to $2^{\Sigma_c \cup \{\lambda\}} \setminus \emptyset$) or not as the reduction holds in any case. □

To compute the most permissive controller (and we will also prove there is one), we build a safety game and solve a safety control problem. It may be necessary to iterate this procedure. Of course, we restrict our attention to TA in the class *dTA* for which the SNNI-VP is decidable.

Let $A = (Q, q_0, X, \Sigma_h \cup \Sigma_l, E, Inv)$ be a TA s.t. $A\backslash\Sigma_h$ is deterministic. The idea of the reduction follows from the following remark: we want to find a controller C s.t. $\mathcal{L}(C(A)\backslash\Sigma_h) = \mathcal{L}(C(A)/\Sigma_h)$. For any controller C we have $\mathcal{L}(C(A)\backslash\Sigma_h) \subseteq \mathcal{L}(C(A)/\Sigma_h)$ because each run of $C(A)\backslash\Sigma_h$ is a run of $C(A)/\Sigma_h)$. To ensure SNNI we must have $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$: indeed, $A\backslash\Sigma_h$ is the largest language that can be generated with no Σ_h actions, so a necessary condition for enforcing SNNI is $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$. The controller $C(A)$ indicates what must be pruned out in A to ensure the previous inclusion. Our algorithm thus proceeds as follows: we first try to find a controller C^1 which ensures that $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$. If $\mathcal{L}(C^1(A)/\Sigma_h) = \mathcal{L}(A\backslash\Sigma_h)$ then C^1 is the most permissive controller that enforces SNNI. It could be that what we had to prune out to ensure $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$ does not render $C^1(A)$ SNNI. In this case we may have to iterate the previous procedure on the new system $C^1(A)$.

We first show how to compute C^1. As $A\backslash\Sigma_h$ is deterministic, we can construct $A_2 = (Q \cup \{q_{bad}\}, q_0^2, X_2, \Sigma_h \cup \Sigma_l, E_2, Inv_2)$ which is a copy of A (with clock renaming) with q_{bad} being a fresh location and s.t. A_2 is a *complete* (*i.e.*, $\mathcal{L}(A_2) = T\Sigma^*$) version of $A\backslash\Sigma_h$ (A_2 is also deterministic). We write *last*$_2(w)$ the state (q, v) reached in A_2 after reading a timed word $w \in T\Sigma^*$. A_2 has the property that $w \in \mathcal{L}(A\backslash\Sigma_h)$ if the state reached in A_2 after reading w is not in *Bad* with *Bad* $= \{(q_{bad}, v) \mid v \in \mathbb{R}_+^X\}$.

Fact 1 *Let $w \in T\Sigma^*$. Then $w \notin \mathcal{L}(A\backslash\Sigma_h) \iff$ last$_2(w) \in$ Bad.*

We now define the product $A_p = A \times_{\Sigma_l} A_2$ and the set of bad states, *Bad*$^\otimes$ of A_p to be the set of states where A_2 is in *Bad*. $\rightarrow_p$ denotes the transition relation of the semantics of A_p and s_p^0 the initial state of A_p. When it is clear from the context we omit the subscript p in $\rightarrow_p$.

Lemma 1. *Let $w \in \mathcal{L}(A)$. Then there is a run $\rho \in$ Runs(A_p) s.t. $\rho = s_p^0 \xrightarrow{w}_p s$ with $s \in$ Bad$^\otimes$ iff $\pi_{\Sigma_l}(w) \notin \mathcal{L}(A\backslash\Sigma_h)$.*

The proof follows easily from Fact 1. Given a run ρ in *Runs*(A_p), we let $\rho_{|1}$ be the projection of the run ρ on A (uniquely determined) and $\rho_{|2}$ be the unique run[4] in A_2 whose trace is $\pi_{\Sigma_l}($*trace*$(\rho))$. The following Theorem proves that any controller C s.t. $C(A)$ is SNNI can be used to ensure that *Bad*$^\otimes$ is not reachable in the game A_p:

Lemma 2. *Let C be a controller for A s.t. $C(A)$ is SNNI. Let $C^\otimes$ be a controller on A_p defined by $C^\otimes(\rho') = C(\rho'_{|1})$. Then, Reach$(C^\otimes(A_p)) \cap$ Bad$^\otimes = \emptyset$.*

Proof. First $C^\otimes$ is well-defined because $\rho'_{|1}$ is uniquely defined. Let C be a controller for A s.t. $C(A)$ is SNNI. Assume *Reach*$(C^\otimes(A_p)) \cap$ *Bad*$^\otimes \neq \emptyset$. By definition, there is a run ρ' in *Runs*$(C^\otimes(A_p))$ such that:

$$\rho' = ((q_0, q_0^2), (\mathbf{0}, \mathbf{0})) \xrightarrow{e_1} ((q_1, q_1'), (v_1, v_1')) \xrightarrow{e_2} \cdots \xrightarrow{e_n} ((q_n, q_n'), (v_n, v_n')) \xrightarrow{e_{n+1}} ((q_{n+1}, q_{n+1}'), (v_{n+1}, v_{n+1}'))$$

with $((q_{n+1}, q_{n+1}'), (v_{n+1}, v_{n+1}')) \in$ *Bad*$^\otimes$ and we can assume $(q_i', v_i') \notin$ *Bad* for $1 \leq i \leq n$ (and $q_0^2 \notin$ *Bad*). Let $\rho = \rho'_{|1}$ and $w = \pi_{\Sigma_l}($*trace*$(\rho')) = \pi_{\Sigma_l}($*trace*$(\rho))$. We can prove (1): $\rho \in$ *Runs*$(C(A))$ 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 $w \in \mathcal{L}(C(A)/\Sigma_h) \setminus \mathcal{L}(C(A)\backslash\Sigma_h)$ *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 which is a contradiction. Hence *Reach*$(C^\otimes(A_p)) \cap$ *Bad*$^\otimes = \emptyset$. □

If we have a controller which solves the safety game $(A_p,$ *Bad*$^\otimes)$, we can build a controller which ensures that $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$. Notice that as emphasized before, this does not necessarily ensure that $C(A)$ is SNNI.

Lemma 3. *Let $C^\otimes$ be a controller for A_p s.t. Reach$(C^\otimes(A_p)) \cap$ Bad$^\otimes = \emptyset$. Let $C(\rho) = C^\otimes(\rho')$ if $\rho'_{|1} = \rho$. C is well-defined and $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$.*

[4] Recall that A_2 is deterministic.

Proof. Let $\rho = (q_0, \mathbf{0}) \xrightarrow{e_1} (q_1, v_1) \xrightarrow{e_2} \cdots \xrightarrow{e_n} (q_n, v_n)$ be a run of A. Since A_2 is deterministic and complete there is exactly one run $\rho' = ((q_0, q_0), (\mathbf{0}, \mathbf{0})) \xrightarrow{e_1} ((q_1, q_1'), (v_1, v_1')) \xrightarrow{e_2} \cdots \xrightarrow{e_n} ((q_n, q_n'), (v_n, v_n'))$ in A_p s.t. $\rho'_{|1} = \rho$. So C is well-defined. Now, assume there is some $w \in \mathcal{L}(C(A)/\Sigma_h) \setminus \mathcal{L}(A\backslash\Sigma_h)$. Then, there is a run ρ in $\mathit{Runs}(C(A)) \subseteq \mathit{Runs}(A)$ s.t. $\pi_{\Sigma_l}(\mathit{trace}(\rho)) = w$, there is a unique run $\rho \in \mathit{Runs}(A_p)$ s.t. $\rho'_{|1} = \rho$ and $\mathit{trace}(\rho') = w$. First by Lemma 1, $\mathit{last}(\rho') \in \mathit{Bad}^{\otimes}$. Second, this run ρ' is in $\mathit{Runs}(C^{\otimes}(A_p))$ because of the definition of C. Hence $\mathit{Reach}(C^{\otimes}(A_p)) \cap \mathit{Bad}^{\otimes} \neq \emptyset$ which is a contradiction. □

It follows that if $C^{\otimes}$ is the most permissive controller for A_p then $C(A)$ is a timed automaton (and can be effectively computed) because the most permissive controller for safety timed games is memoryless. More precisely, let $RG(A_p)$ be the the region graph of A_p. C is memoryless on $RG(A_p\backslash\Sigma_h)$ because A_2 is deterministic. The memory required by C is at most $RG(A\backslash\Sigma_h)$ on the rest of the region graph of $RG(A_p)$.

Assume the safety game $(A_p, \mathit{Bad}^{\otimes})$ can be won and $C^{\otimes}$ is the most permissive controller. Let C be the controller obtained using Lemma 3. Controller C ensures that $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$. But as the following example shows, it may be the case that $C(A)$ is not SNNI.

Example 4. Consider the TA K of Figure 5 with $\Sigma_h = \{h\}$ and $\Sigma_c = \{a\}$. We can compute $C(K)$ from $C^{\otimes}$ which satisfies $\mathit{Reach}(C^{\otimes}(K \times_{\Sigma_l} K_2)) \cap \mathit{Bad}^{\otimes} = \emptyset$, and is given by the sub-automaton of K with the plain arrows. $C(K)$ is obviously not SNNI. For the example of $A(1)$ in Figure 1, if we compute C in the same manner, we obtain $C(A(1)) = A(2)$ and moreover $\mathcal{L}(C(A(1))/\Sigma_h) = \mathcal{L}(A(1)\backslash\Sigma_h)$. And then the most permissive sub-system which is SNNI is given by $C(A(1)) = A(2)$ (the guard $x \geq 1$ of $A(1)$ is strenghtened). ■

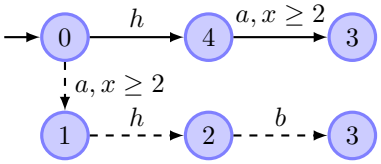

Fig. 5. The Automaton K

The example of Figure 5 shows that computing the most permissive controller on A_p is not always sufficient. Actually, we may have to iterate the computation of the most permissive controller on the reduced system $C(A)$.

Lemma 4. *Consider the controller C as defined in Lemma 3. If $C(A)\backslash\Sigma_h \approx A\backslash\Sigma_h$ then $C(A)$ is SNNI.*

Proof. If $C(A)\backslash\Sigma_h \approx A\backslash\Sigma_h$, then, $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h) = \mathcal{L}(C(A)\backslash\Sigma_h)$. As $\mathcal{L}(C(A)\backslash\Sigma_h) \subseteq \mathcal{L}(C(A)/\Sigma_h)$ is always true, $\mathcal{L}(C(A)/\Sigma_h) = \mathcal{L}(C(A)\backslash\Sigma_h)$ and so, $C(A)$ is SNNI. □

Let $\perp$ be the symbol that denotes non controllability (or the non existence of a controller). We inductively define the sequence of controllers C^i and timed automata A^i as follows:

– let C^0 be the controller defined by $C^0(\rho) = 2^{\Sigma_c \cup \{\lambda\}}$ and $A^0 = C^0(A) = A$;

– Let $A_p^i = A^i \times_{\Sigma_l} A_2^i$ and $C_{i+1}^{\otimes}$ be the most permissive controller for the safety game $(A_p^i, \mathit{Bad}_i^{\otimes})$ ($\perp$ if no such controller exists). We use the notation $\mathit{Bad}_i^{\otimes}$ because this set depends on A_2^i. We define C^{i+1} using Lemma 3: $C^{i+1}(\rho) = C_{i+1}^{\otimes}(\rho')$ if $\rho'_{|1} = \rho$. Let $A^{i+1} = C^{i+1}(A^i)$.

By Lemma 4, if $C^{i+1}(A^i)\backslash\Sigma_h \approx A^i\backslash\Sigma_h$ then $C^{i+1}(A^i)$ is SNNI. Therefore this condition is a sufficient condition for the termination of the algorithm defined above:

Lemma 5. *There exists an index $i \geq 1$ s.t. $C^i(A^{i-1})$ is SNNI or $C^i = \perp$.*

Proof. We prove that the region graph of $C^{i+1}(A^i)$ is a subgraph of the region graph of $C^1(A^0)$ for $i \geq 1$. By Lemma 3 (and the remark following it), $C^1(A^0)$ is a subgraph of $RG(A \times A_2)$. Moreover C^1 is memoryless on $A\backslash\Sigma_h$ and requires a memory of less than $|RG(A\backslash\Sigma_h)|$ on the remaining part. Assume on this part, a node of $RG(A \times A_2)$ is of the form $((q, r), k)$ where q is a location of A and r a region of A and $k \in \{1, |RG(A\backslash\Sigma_h)|\}$.

Assume $RG(A^k)$ is a subgraph of $RG(A^{k-1})$ for $k \geq 2$ and $RG(A^{k-1}\backslash\Sigma_h)$ is subgraph of $RG(A\backslash\Sigma_h)$. Using Lemma 3, we can compute $A^k = C^k(A^{k-1})$ and: (1) $RG(A^k\backslash\Sigma_h)$ is a subgraph of $A^{k-1}\backslash\Sigma_h$ and (2) the memory needed for $C_k^{\otimes}$ on the remaining part is less than $|RG(A^{k-1})|$. Actually, because $A^{k-1}\backslash\Sigma_h$ is deterministic, no more memory is required for C^k. Indeed, the memory corresponds to the nodes of $A^k\backslash\Sigma_h$. Thus a node of $RG(A^k)$ which is not in $RG(A^k\backslash\Sigma_h)$ is of the form $((q, r), k, k')$ with $k = k'$ or $k' = q_{bad}$. This implies that $RG(A^k)$ is a subgraph of $RG(A^{k-1})$.

The most permissive controller $C_i^{\otimes}$ will either disable at least one controllable transition of $A_p^{i-1}\backslash\Sigma_h$ or keep all the controllable transitions of $A_p^{i-1}\backslash\Sigma_h$. In the latter case $A^i\backslash\Sigma_h = A^{i-1}\backslash\Sigma_h$ and otherwise $|RG(A^i\backslash\Sigma_h)| < |RG(A^{i-1}\backslash\Sigma_h)|$. This can go on at most $|RG(A\backslash\Sigma_h)|$ steps. In the end either $A^i\backslash\Sigma_h = A^{i-1}\backslash\Sigma_h$ and this implies that $A^i\backslash\Sigma_h \approx A^{i-1}\backslash\Sigma_h$ (Lemma 4) or it is impossible to control A^{i-1} and $C^i = \perp$. In any case, our algorithm terminates in less than $|RG(A)|$ steps. □

To prove that we obtain the most permissive controller which enforces SNNI, we use the following Lemma:

Lemma 6. *If M is a controller such that $\mathcal{L}(M(A)/\Sigma_h) = \mathcal{L}(M(A)\backslash\Sigma_h)$, then $\forall i \geq 0$ and $\forall \rho \in \mathit{Runs}(A)$, $M(\rho) \subseteq C^i(\rho)$.*

Proof. The proof is by induction:

– for $i = 0$ it holds trivially.
– Assume the Lemma holds for indices up until i. Thus we have $\mathit{Runs}(M(A)) \subseteq \mathit{Runs}(A^i)$. Therefore, we can define M over A^i and $M(A^i)$ is SNNI. By Lemma 2, $M^{\otimes}$ is a controller for the safety game $(A_p^i, \mathit{Bad}_i^{\otimes})$, therefore $M^{\otimes}(\rho') \subseteq C_{i+1}^{\otimes}(\rho')$ because $C_{i+1}^{\otimes}$ is the most permissive controller. This implies that $M(\rho) \subseteq C^{i+1}(\rho)$ by definition of C^{i+1}. □

Using Lemma 5, the sequence C^i converges to a fix-point. Let C^* denote this fix-point.

Lemma 7. *C^* is the most permissive controller for the SNNI-CSP.*

Proof. Either $C^* = \bot$ and there is no way of enforcing SNNI (Lemma 2), or $C^* \neq \bot$ is such that $\mathcal{L}(C^*(A)/\Sigma_h) = \mathcal{L}(C^*(A)\backslash\Sigma_h)$ by Lemma 3. As for any valid controller M such that $\mathcal{L}(M(A)/\Sigma_h) = \mathcal{L}(M(A)\backslash\Sigma_h)$ we have $M(\rho) \subseteq C^*(\rho)$ for each $\rho \in$ *Runs*(A) (Lemma 6) the result follows. □

Lemma 5 proves the existence of a bound on the number of times we have to solve safety games. For a timed automaton A in *dTA*, let $|A|$ be the size of A.

Lemma 8. *For a dTA A, C^* can be computed in $O(2^{4.|A|})$.*

Proof. As the proof of Lemma 5 shows, the region graph of A^i is a subgraph of the region graph of A^1, $\forall i \geq 1$, and the algorithm ends in less than $|RG(A)|$ steps. Computing the most permissive controller for A^i_p avoiding $\mathit{Bad}^{\otimes}_i$ can be done in linear time in the size of the region graph of A^i_p. As $RG(A^i)$ is a subgraph of $RG(A^1)$, $RG(A^i_p)$ is a subgraph of $RG(A^1_p)$. So we have to solve at most $|RG(A)|$ safety games of sizes at most $|RG(A^1_p)|$. As A^1 is a subgraph of $A^0_p = A^0 \times_{\Sigma_l} A^0_2$, $|RG(A^1)| \leq |RG(A)|^2$. And as $A^1_p = A^1 \times_{\Sigma_l} A^1_2$, $|RG(A^1_p)| \leq |RG(A)|^3$. So, C^* can be computed in $O(|RG(A)|.|RG(A^1_p)|) = O(|RG(A)|^4) = O(2^{4.|A|})$. □

Theorem 4. *For dTA, the SNNI-CP and SNNI-CSP are EXPTIME-complete.*

For the special case of finite automata we even have:

Lemma 9. *For finite automata $C^* = C^2$.*

Proof. We know that $\mathcal{L}(C^2(A)\backslash\Sigma_h) \subseteq \mathcal{L}(C^1(A)\backslash\Sigma_h)$. Suppose that $\exists w$ s.t. $w \in \mathcal{L}(C^1(A)\backslash\Sigma_h)$ and $w \notin \mathcal{L}(C^2(A)\backslash\Sigma_h)$ (w cannot not be the empty word). We can assume that $w = u.l$ with $u \in \Sigma_l^*$, $l \in \Sigma_l \cap \Sigma_c$ and $u \in \mathcal{L}(C^1(A)\backslash\Sigma_h)$ and $u.l \notin \mathcal{L}(C^2(A)\backslash\Sigma_h)$ (l is the first letter which witnesses the non membership property). If l had to be pruned in the computation of C^2, it is because there is a word $u.l.m$ with $m \in \Sigma_u^*$ s.t. $\pi_{\Sigma_l}(u.l.m) \in \mathcal{L}(C^1(A)/\Sigma_h)$ but $\pi_{\Sigma_l}(u.l.m) \notin \mathcal{L}(C^1(A)\backslash\Sigma_h)$. But by definition of C^1, $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$ (Lemma 3) and thus $\pi_{\Sigma_l}(u.l.m) \in \mathcal{L}(A\backslash\Sigma_h)$. As $u.l \in \Sigma_l^*$, $\pi_{\Sigma_l}(u.l.m) = u.l.\pi_{\Sigma_l}(m)$ and $\pi_{\Sigma_l}(m) \in \Sigma_u^*$. Since $u.l \in \mathcal{L}(C^1(A)\backslash\Sigma_h)$ and $\pi_{\Sigma_l}(m) \in \Sigma_u^*$, we have $u.l.\pi_{\Sigma_l}(m) \in \mathcal{L}(C^1(A)\backslash\Sigma_h)$ which is a contradiction. Thus $\mathcal{L}(C^2(A)\backslash\Sigma_h) = \mathcal{L}(C^1(A)\backslash\Sigma_h)$ which is our stoping condition by lemma 4 and thus $C^* = C^2$. □

It follows that when $A\backslash\Sigma_h$ is deterministic, A a finite automaton, the SNNI-CSP is PSPACE-complete. As untimed automata can always be determinized, we can extend our algorithm to untimed automata when $A\backslash\Sigma_h$ non-deterministic. It suffices to determinize $A^i_2, i = 1, 2$:

Theorem 5. *For a finite automaton A such that $A\backslash\Sigma_h$ is non deterministic, the SNNI-CSP can be solved in EXPTIME.*

Proposition 1. *There is a family of FA $(A_i)_{i\geq 0}$ such that: (i) there is a most permissive controller D_i^* s.t. $D_i^*(A_i)$ is SNNI and (ii) the memory required by D_i^* is exponential in the size of A_i.*

Proof. Let A be a finite automaton over the alphabet Σ. Define the automaton A' as given by Figure 6. Assume the automaton B is the sub-automaton of A' with initial state q'_0. We take $\Sigma_h = \{h\} = \Sigma_u$ and $\Sigma_l = \Sigma = \Sigma_c$. The most permissive controller D s.t. $D(A')$ is SNNI generates the largest sub-language of $\mathcal{L}(A')$ s.t. $\mathcal{L}(A'\backslash\Sigma_h) = \mathcal{L}(A'/\Sigma_h)$ and thus it generates $\mathcal{L}(A) = \mathcal{L}(A'\backslash\Sigma_h)$.

The controller D is memoryless on $A'\backslash\Sigma_h$ as emphasized in Lemma 3. It needs finite memory on the remaining part *i.e.*, on B. The controller D on B gives for each run a set of events of Σ that can be enabled: $D(q_0 \xrightarrow{h} q'_0 \xrightarrow{w} q'_0) = X$ with $w \in \Sigma^*$ and $X \subseteq \Sigma_l$.As B is deterministic, D needs only the knowledge of w and we can write $D(hw)$ ignoring the states of A'. For B we can even write $D(w)$ instead of $D(hw)$. Define the equivalence relation $\equiv$ on Σ^* by: $w \equiv w'$ if $D(w) = D(w')$. Denote the class of a word w by $[w]$. Because D is memory bounded, $\Sigma^*_{/\equiv}$ is of finite index which is exactly the memory needed by D.

Thus we can define an automaton $D_{/\equiv} = (M, m_0, \Sigma, \rightarrow)$ by: $M = \{[w] \mid w \in \Sigma^*\}$, $m_0 = [\varepsilon]$, and $[w] \xrightarrow{a} [wa]$ for $a \in D(hw)$. $D_{/\equiv}$ is an automaton which accepts $\mathcal{L}(A)$ (and it is isomorphic to $D(B)$) and the size of which is the size of D because B has only one state. This automaton is deterministic and thus $D_{/\equiv}$ is also deterministic and accepts $\mathcal{L}(A)$. There is a family $(A_i)_{i\geq 0}$ of non-deterministic FA, such that the deterministic and language-equivalent automaton of each A_i requires at least exponential size. For each of these A_i we construct the controller $D^i_{/\equiv}$ as described before, and this controller must have at least an exponential size (w.r.t. to A_i). This proves the EXPTIME lower bound. □

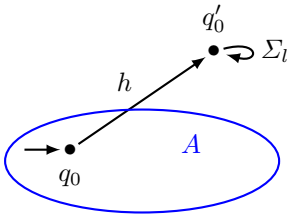

Fig. 6. Automaton B

5 Conclusion and Future Work

In this paper we have studied the strong non-deterministic non-interference control problem (SNNI-CP) and control synthesis problem (SNNI-CSP) in the timed setting. The main results we have obtained are: (1) the SNNI-CP can be solved if $A\backslash\Sigma_h$ can be determinized and is undecidable otherwise; (2) the SNNI-CSP can be solved by solving a finite sequence of safety games if $A\backslash\Sigma_h$ can be determinized.

	A Timed Automaton		A Finite Automaton	
	$A\backslash\Sigma_h$ Non-Det.	$A\backslash\Sigma_h$ Det.	$A\backslash\Sigma_h$ Non-Det.	$A\backslash\Sigma_h$ Det.
SNNI-VP	undecidable	PSPACE-C	PSPACE-C	PTIME
SNNI-CP	undecidable	EXPTIME-C	PSPACE-C	PTIME
SNNI-CSP	undecidable	EXPTIME-C	EXPTIME	PSPACE-C

Table 2. Summary of the Results

The summary of the results is given in Table 2. For non-deterministic FA, we have proved that EXPTIME is a lower bound.

Our future work will consist in extending this work to other types of non-interference properties, CSNNI and BSNNI which are more involved than SNNI. Indeed, there is not always a least restrictive controller for (bi)simulation based non-interference. Thus it is interesting to determine conditions under which a least restrictive controller exists for the BSNNI-CSP and CSNNI-CSP.

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 LNCS, Springer (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. Focardi, R., Ghelli, A., Gorrieri, R.: Using non interference for the analysis of security protocols. Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols (1997).
4. van der Meyden, R., Zhang, C.: Algorithmic verification of noninterference properties. Elec. Notes in Theo. Comp. Science **168**(1) (2006) 61–75. Proceedings of the Second International Workshop on Views on Designing Complex Architectures (VODCA 2006).
5. D'Souza, D., Raghavendra, K.R., Sprick, B.: An automata based approach for verifying information flow properties. Elec. Notes in Theo. Comp. Science **135** (2005) 39–58. Proceedings of the Second Workshop on Automated Reasoning for Security Protocol Analysison (ARSPA'05).
6. Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications **21**(1) (2003).
7. Gardey, G., Mullins, J., Roux, O.H.: Non-interference control synthesis for security timed automata. Elec. Notes in Theo. Comp. Science **180**(1) (2005) 35–53. Proceedings of the 3rd International Workshop on Security Issues in Concurrency (SecCo'05).
8. Cassez, F., Mullins, J., Roux, O.H.: Synthesis of non-interferent systems. In: Proceedings of the 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, Springer (2007) 307–321.
9. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235.
10. Finkel, O.: On decision problems for timed automata. Bulletin of the European Association for Theoretical Computer Science **87** (2005) 185–190.
11. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science Munich (STACS '95). Volume 900 of LNCS, Springer (1995) 229–242.
12. D'Souza, D., Madhusudan, P.: Timed control synthesis for external specifications. In: proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science Munich (STACS '02). Volume 2285 of LNCS, Springer (2002) 571–582.
13. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time: Preliminary report. In: STOC, ACM (1973) 1–9.
14. Henzinger, T., Kopke, P.: Discrete-time control for rectangular hybrid automata. In: Proceedings of the 24th International Colloquium on Automata, Languages, and Programming (ICALP '97). Volume 1256 of LNCS, Springer (1997) 582–593