International Journal of Control, 2014
http://dx.doi.org/10.1080/00207179.2014.944356

Control and synthesis of non-interferent timed systems

Gilles Benattar[a], Franck Cassez[b], Didier Lime[c,*] and Olivier H. Roux[c]

[a]*ClearSy, Paris, France;* [b]*National ICT Australia, Sydney, Australia;* [c]*École Centrale de Nantes, IRCCyN UMR CNRS 6597, Nantes, France*

(*Received 10 July 2013; accepted 9 July 2014*)

We focus on the control and the synthesis of secure timed systems which are modelled as timed automata. The security property that the system must satisfy is a *non-interference* property. Intuitively, non-interference ensures the absence of any causal dependency from a high-level domain to a lower level domain. 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 two (bi)simulation-based variants thereof (cosimulation-based SNNI and bisimulation-based SNNI). These properties and their extensions have been mostly studied in the context of discrete event systems, while it is now well-known that time is an important attack vector against secure systems.

At the same time, there is an obvious interest in going beyond simple verification to control problems: to be able to automatically make systems secure.

We consider non-interference properties in the challenging setting of control of dense-time systems specified by *timed automata* and we study the two following problems: (1) check whether it is possible to find a sub-system so that it is non-interferent; if yes, (2) compute a (largest) sub-system which is non-interferent.

We exhibit decidable sub-classes for these problems, assess their theoretical complexities and provide effective algorithms based on the classical framework of timed games.

Keywords: control; synthesis; non-interference; timed automaton; safety timed games

1. Introduction

Modern computing environments allow the use of programmes that are sent or fetched from different sites. Such programmes may deal with secret information such as private data (of a user) or classified data (of an organisation). While operating systems provide isolation through separate memory spaces, processes still interact via files, pipes, network connections, shared memory, and other mechanisms. Moreover, by measuring the time required by certain operations, an attacker can learn information about the past activities and time is a potential attack vector against secure systems.

One of the basic concerns in such a context is to ensure that the programmes 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 behaviours.

Non-interference: We assume that there are two privileged levels, 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 we focus on are strong non-deterministic non-interference (SNNI), cosimulation-based strong non-deterministic non-interference (CSNNI), and bisimulation-based strong non-deterministic non-interference (BSNNI). The *non-interference verification problem*, for a given system S, consists of checking whether S is non-interferent. This notion is formalised by Rushby, 1992, in terms of input–output automata and, in the same paper, extended to intransitive non-interference (INI). INI enables the specification of a generalised class of security policies dealing with channel control mechanisms. In Hadj-Alouane, Lafrance, Lin, Mullins, and Yeddes (2005a, 2005b), INI is formalised in the setting of discrete event systems (DES). Opacity (Mazaré, 2004) is a more general notion where different observation functions are compared with respect to their power of discovering secret (or opaque) information. Opacity is expressive enough to define SNNI but is not comparable to BSNNI. It is worth noticing that non-interference properties are out of the scope of the common safety/liveness classification of system properties (Focardi & Gorrieri, 2001).

Verification of non-interference: Verification of information-flow security properties (Focardi & Gorrieri,

*Corresponding author. Email: Didier.Lime@ec-nantes.fr

1997, 2001) can be applied to the analysis of cryptographic protocols where many uniform and concise characterisations 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 insecure by defining the security property using SNNI (Focardi, Ghelli, & Gorrieri, 1997), and other examples of the use of non-interference in computer systems and protocols for checking security properties can be found in Bossi, Piazza, and Rossi (2007), Barthe, Pichardie, and Rezk (2007), Kammuller (2008), and Krohn and Tromer (2009). There is a large body of works on the use of static analysis techniques to guarantee information-flow policies. A general overview can be found in Sabelfeld and Myers (2003). In van der Meyden and Zhang (2006), the authors consider the complexity of many non-interference *verification* problems. In D'Souza, Raghavendra, and Sprick (2005), an exponential time-decision procedure for checking whether a finite-state system satisfies a given basic security predicate (BSP) is presented. The problem of the verification of INI is addressed in Hadj-Alouane et al. (2005a, 2005b) using algorithmic approaches in the DES setting.

Control for non-interference: In case a system is not non-interferent, it is interesting to investigate how it can be rendered non-interferent.

This is the scope of this paper where we consider the problem of *synthesising* non-interferent timed systems. In contrast to verification, the *non-interference synthesis problem* assumes that the system is *open*, i.e. we can restrict the behaviours of S: some events, from a particular set $\Sigma_c \subseteq \Sigma_l \cup \Sigma_h$, 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?' where $C(S)$ is 'S controlled by C'. The associated *synthesis problem* asks to compute a witness controller C when one exists.

As mentioned earlier, SNNI is expressive enough, for example, to prove that the Needham–Schroeder protocol is flawed (Focardi et al., 1997). Controller synthesis enables one to find automatically the patch(es) to apply to make such a protocol secure.

Recently, supervisory control for opacity property has been studied in Saboori and Hadjicostis (2008) and Cassez, Dubreil, and Marchand (2009, 2012). In Cassez, Mullins, and Roux (2007) the controller synthesis problem for non-interference properties is addressed and in Moez, Lin, and Ben Hadj-Alouane (2009), supervisory control to enforce INI for three-level security systems is proposed in the untimed setting.

The case of timed systems: It is well-known that time is a potential attack vector against secure systems (see e.g. Kocher, 1996; Felten & Schneider, 2000; Bortz & Boneh, 2007; Kotcher, Pei, Jumde, & Jackson, 2013) and an attacker that can measure time as both more powerful and more realistic. It is indeed not surprising that a non-interferent system can become interferent when timing constraints are added (Gardey, Mullins, & Roux, 2005). The analysis of dense-time systems is more complex naturally leading to the question of whether proof techniques developed in the untimed setting can be generalised for timed systems in order to be able to capture, besides the logical information flows, also the *time-dependent* interference. Some untimed bisimulation-based non-interference properties for information flow studied in Focardi and Gorrieri (2001) have been reformulated in R. Focardi and Martinelli (2003) in a discrete time setting. In Barbuti and Tesei (2003), some state-based and trace-based non-interference properties have been introduced in a dense-time setting using timed automata (TA).

It was proved in Gardey et al. (2005) that the problem of the verification of timed SNNI is undecidable for non-deterministic TA. Opacity which is more general than SNNI is also undecidable for timed systems (Cassez, 2009) and thus the associated control problem is undecidable as well. The non-interference synthesis problem for dense-time systems specified by TA was first considered in Gardey et al. (2005). The non-interference property considered in Gardey et al. (2005) is the *state* non-interference property, which is less demanding than the one we consider here. Finally, Benattar, Cassez, Lime, and Roux (2009) address a decidable sub-class of the SNNI control problem for timed systems.

Even though the objective of SNNI control is to restrict the (timed) language of the system, with high-level actions considered unobservable, so that it is equal to the language of the system where these actions are cut of; this problem is not easily reducible to a problem of time control with partial observability as defined in Lin and Wonham (1988), Lin and Wonham (1995), Kupferman and Vardi (1997), Lamouchi and Thistle (2000), D'Souza and Madhusudan (2002), and Bouyer, D'Souza, Madhusudan, and Petit (2003). The reason is that the controller restricts at the same time both the system and the target language. This is discussed in more details in Section 5.4.

Our contribution: In this paper, we address the challenging problem of controlling dense-time systems so that they become non-interferent. We focus on the most basic forms of non-interference as a first step towards the more sophisticated versions, such as INI.

The main contributions of this paper are as follows: (1) we exhibit a class dTA of timed automata for which the SNNI-VP is decidable; (2) We prove that deciding whether there is a controller C for a timed automaton A such that (s.t. in the following) $C(A)$ is SNNI, is decidable for the previous class dTA; (3) we reduce the SNNI controller synthesis problem to solving a sequence of *safety timed games*; (4) we show that there is not always a most permissive controller for CSNNI and BSNNI; (5) we prove that the control problem for CSNNI is decidable for the class dTA and that the CSNNI controller synthesis problem for dTA reduces to the SNNI controller synthesis problem; and (6) we also give the theoretical complexities of these problems.

This paper extends the results of our previous paper (Benattar et al., 2009) as follows : Section 5 is an extension with unpublished proofs of the results of Benattar et al. (2009). All the others results are new.

Organisation of the paper: Section 2 recalls the basics of TA, timed languages, and some results on safety timed games. Section 3 gives the definition of the non-interference properties we are interested in. Section 4 addresses the verification of non-interference properties in the timed setting. Section 5 gives the definition of the non-interference synthesis problem and presents the main result: we show that there is the largest sub-system which is SNNI and this sub-system is effectively computable. Section 6 addresses the control problem and controller synthesis problem for CSNNI and BSNNI properties. Finally, we conclude in Section 7.

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 $\vec{0}_X$ be the valuation s.t. $\vec{0}_X(x) = 0$ for each $x \in X$ and use $\vec{0}$ 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 $x \bowtie c$, with $x \in X$, $\bowtie \in \{<, \leq, =, \geq, >\}$, 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. $(v + t)(x) = v(x) + t$. We 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 wordw* over Σ is a sequence $w = (\delta_0, a_0)(\delta_1, a_1) \dots (\delta_n, a_n)$ 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[1] 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. $(\delta_1, \varepsilon)(\delta_2, a) = (\delta_1 + \delta_2, a)$: this means that languagewise, we can always eliminate the ε action by taking into account its time interval in the next visible action. Given timed words $w \in T\Sigma^*$ and $L \subseteq \Sigma$, the *projection* of w over L is denoted by $\mathbf{proj}_L(w)$ and is defined by $\mathbf{proj}_L(w) = (\delta_0, b_0)(\delta_1, b_1) \dots (\delta_n, b_n)$ 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 $a_0 a_1 \dots a_n$ of Σ^*.

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

Definition 2.1 (Timed transition system (TTS)): A *timed transition system (TTS)* is a tuple $\mathcal{S} = (Q, q_0, \Sigma^\varepsilon, \to)$, where Q is a set of states, q_0 is the initial state, Σ a finite alphabet of actions, and $\to \subseteq Q \times \Sigma^\varepsilon \cup \mathbb{R}_+ \times Q$ is the transition relation. We use the notation $q \xrightarrow{e} q'$ if $(q, e, q') \in \to$. Moreover, TTS should satisfy the classical time-related conditions where $d, d' \in \mathbb{R}_{\geq 0}$: (i) time determinism: $(q \xrightarrow{d} q') \wedge (q \xrightarrow{d} q'') \Rightarrow (q' = q'')$, ii) time additivity: $(q \xrightarrow{d} q') \wedge (q' \xrightarrow{d'} q'') \Rightarrow (q \xrightarrow{d+d'} q'')$, (iii) null delay: $\forall q : q \xrightarrow{0} q$, and (iv) time continuity: $(q \xrightarrow{d} q') \Rightarrow (\forall d' \leq d, \exists q'', q \xrightarrow{d'} q'')$.

A run ρ of $\mathcal{S}$ from q_0 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. $(q_i, e_i, q_{i+1}) \in \to$ for $0 \leq i \leq n - 1$. We denote by $last(\rho)$ the last state of the sequence, i.e. the state q_n. We let $Runs(q, \mathcal{S})$ be the set of runs from q in $\mathcal{S}$ and $Runs(\mathcal{S}) = Runs(q_0, \mathcal{S})$. We write $q \overset{\varepsilon}{\Rightarrow} q'$, if there is a run $q \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} q'$ from q to q', i.e. $\overset{\varepsilon}{\Rightarrow} \overset{\text{def}}{=} (\xrightarrow{\varepsilon})*$. Given $a \in \Sigma \cup \mathbb{R}_+$, we define $\overset{a}{\Rightarrow} \overset{\text{def}}{=} \overset{\varepsilon}{\Rightarrow}\xrightarrow{a}\overset{\varepsilon}{\Rightarrow}$. 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 $Reach(\mathcal{S}) = \{q \mid q_0 \xrightarrow{*} q\}$. 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 $trace(\rho) = (\delta_0, e_0)(\delta_1, e_1) \dots (\delta_n, e_n)$.

Definition 2.2 (Timed automata (TA)): 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 $x \bowtie c$ with $\bowtie \in \{<, \leq\}$.

A finite (or untimed) automaton $A = (Q, q_0, \Sigma^\varepsilon, E)$ is a special kind of timed automaton with $X = \varnothing$, 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 TA cannot always be determinised (i.e. find a deterministic TA which accepts the same language as a non-deterministic one, see Alur & Dill, 1994), and moreover, checking whether a TA determinisable is undecidable (Finkel, 2005).

Definition 2.3 (Semantics of timed automata): The *semantics* of a timed automaton $A = (Q, q_0, X, \Sigma^\varepsilon, E, Inv)$ is the TTS $\mathcal{S}^A = (S, s_0, \Sigma^\varepsilon, \to)$ with $S = Q \times (\mathbb{R}^+)^X$, $s_0 = (q_0, \vec{0})$, and $\to$ defined as follows:

$$(q, v) \xrightarrow{a} (q', v') \text{ iff } \exists (q, \gamma, a, R, q') \in E \text{ such that } \begin{cases} \gamma(v) = \mathsf{true} \\ v' = v[R \mapsto 0] \\ Inv(q')(v') = \mathsf{true} \end{cases}$$

$$(q, v) \xrightarrow{\delta} (q, v') \text{ iff } \begin{cases} v' = v + \delta \\ \forall \delta', 0 \leq \delta' \leq \delta, \\ Inv(q)(v + \delta') = \mathsf{true} \end{cases}$$

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 $w = trace(\rho)$ for some $\rho \in Runs(A)$. The timed language generated by A, $\mathcal{L}(A)$ is the set of timed words generated by A.

Definition 2.4 (Language equivalence): Two automata A and B are *language equivalent*, denoted by $A \approx_{\mathcal{L}} B$, if $\mathcal{L}(A) = \mathcal{L}(B)$, i.e. they generate the same set of timed words.

Definition 2.5 (Simulation): Let $\mathcal{T}_1 = (S_1, s_0^1, \Sigma^\varepsilon, \rightarrow_1)$, $\mathcal{T}_2 = (S_2, s_0^2, \Sigma^\varepsilon, \rightarrow_2)$ be two TTS. Let $\mathcal{R} \subseteq S_1 \times S_2$ be a relation, s.t. $\mathcal{R}$ is total for S_2. $\mathcal{R}$ is a weak simulation of $\mathcal{T}_2$ by $\mathcal{T}_1$ iff:

(1) $s_0^1 \mathcal{R} s_0^2$,
(2) $\forall (s, p) \in S_1 \times S_2$, such that $s\mathcal{R}p$:

- If $p \xrightarrow{\varepsilon_2} p'$ then $\exists s'$ such that $s \xrightarrow{\varepsilon_1} s'$ and $s'\mathcal{R}p'$,
- $\forall a \in \Sigma \cup \mathbb{R}_+$, if $p \xrightarrow{a_2} p'$ then $\exists s'$ such that $s \xrightarrow{a_1} s'$ and $s'\mathcal{R}p'$.

$\mathcal{T}_1$ weakly simulates $\mathcal{T}_2$ if there exists a weak simulation $\mathcal{R}$ of $\mathcal{T}_2$ by $\mathcal{T}_1$ and we note $\mathcal{T}_1 \sqsubseteq_{\mathcal{W}} \mathcal{T}_2$. Let A_1 and A_2 be two TA, we say that A_1 weakly simulates A_2 if the semantics of A_1 weakly simulates the semantics of A_2, and we note $A_1 \sqsubseteq_{\mathcal{W}} A_2$.

Definition 2.6 (Cosimulation): Two timed automata A_1 and A_2 are *cosimilar* iff $A_1 \sqsubseteq_{\mathcal{W}} A_2$ and $A_2 \sqsubseteq_{\mathcal{W}} A_1$. We note $A_1 \approx_{\mathcal{CW}} A_2$.

Definition 2.7 (Bisimulation): Two timed automata A_1 and A_2 are *bisimilar* iff there exists a simulation $\mathcal{R}$ of A_2 by A_1 such that $\mathcal{R}^{-1}$ is a weak simulation of A_1 by A_2. We note $A_1 \approx_{\mathcal{W}} A_2$.

Note that when no ε transition exists, we obtain *strong* versions of similarity and bisimilarity.

Definition 2.8 (Product of timed automata): 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 $X_1 \cap X_2 = \varnothing$. Let $\Sigma_a \subseteq \Sigma$. The *synchronised product* of A_1 and A_2, with respect to Σ_a, is the timed automaton $A_1 \times_{\Sigma_a} A_2 = (Q_1 \times Q_2, (q_{01}, q_{02}), X_1 \cup X_2, \Sigma^\varepsilon, E, Inv)$ where E is defined as follows:

- $((q_1, q_2), \gamma_1 \wedge \gamma_2, a, R_1 \cup R_2, (q_1', q_2')) \in E$ if $a \in \Sigma_a$, $(q_1, \gamma_1, a, R_1, q_1') \in E_1$ and $(q_2, \gamma_2, a, R_2, q_2') \in E_2$;
- $((q_1, q_2), \gamma, a, R, (q_1', q_2')) \in E$ if $a \in \Sigma \setminus \Sigma_a$ and $\begin{cases} (q_1, \gamma, a, R, q_1') \in E_1 \text{ and } q_2' = q_2 \\ \text{or } (q_2, \gamma, a, R, q_2') \in E_2 \text{ and } q_1' = q_1 \end{cases}$

and where $Inv((q_1, q_2)) = Inv_1(q_1) \wedge Inv_2(q_2)$.

It means that synchronisation occurs only for actions in Σ_a. When it is clear from the context, we omit the subscript Σ_a in $\times_{\Sigma_a}$.

Moreover, 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 labelled 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 Maler, Pnueli, and Sifakis (1995).

Definition 2.9 (Timed game automaton (TGA)): A *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 as $\cup_{1 \leq i \leq k} (\ell_i, Z_i)$, 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 to decide whether 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 $last(\rho) \xrightarrow{\delta} (q', v')$ for some $\delta > 0$. A controller C is *state-based* or *memoryless* whenever $\forall \rho, \rho' \in Runs(A)$, $last(\rho) = last(\rho')$ implies that $C(\rho) = C(\rho')$.

Remark 1: We assume that a controller gives a set of actions that are enabled which differs from standard definitions (Maler et al., 1995) 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:

- $(q_0, \vec{0}) \in Runs(C(A))$,

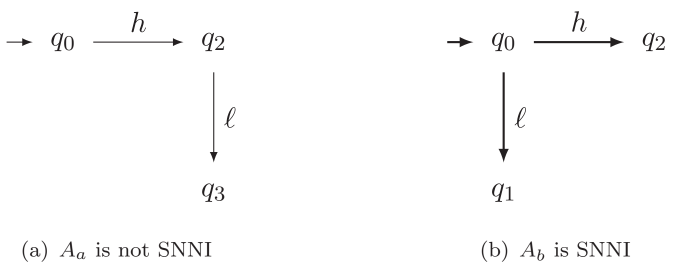


Figure 1. Examples for the SNNI property.

- if $\rho \in Runs(C(A))$ and $\rho \xrightarrow{e} s' \in Runs(A)$, then $\rho \xrightarrow{e} s' \in Runs(C(A))$ 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.* $0 \leq \delta < e$, $last(\rho) \xrightarrow{\delta} last(\rho) + \delta \wedge \lambda \in C(\rho \xrightarrow{\delta} last(\rho) + \delta)$.

$C(A)$ 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 $Reach(C(A)) \cap Bad = \varnothing$. For safety timed games, the results are the following (D'Souza & Madhusudan, 2002; Maler et al., 1995):

- it is EXPTIME-complete to decide 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. Formal definitions of non-interference properties

In the sequel, we will consider TA defined on a set of actions $\Sigma = \Sigma_l \cup \Sigma_h$ with $\Sigma_l \cap \Sigma_h = \varnothing$, where Σ_h are the *high-level* actions and Σ_l the *low-level* actions. In order to define the different classes of non-interference properties on an automaton A, we are going to compare $A \backslash \Sigma_h$ and A / Σ_h with respect to different criteria.

3.1 *Strong non-deterministic non-interference (SNNI)*

The property SNNI has been introduced by Focardi and Gorrieri (2001) as a *trace-based* generalisation of non-interference for concurrent systems. SNNI has been extended to timed models in Gardey et al. (2005).

Definition 3.1: A timed automaton A is SNNI iff $A \backslash \Sigma_h \approx_{\mathcal{L}} A / \Sigma_h$.

Since finite automata are TA with no clocks, the definition also applies to finite automata.

Moreover, as $\mathcal{L}(A \backslash \Sigma_h) \subseteq \mathcal{L}(A / \Sigma_h)$, we can give a simple characterisation of the SNNI property.

Proposition 3.2: *A timed automaton A, is SNNI, iff* $\mathcal{L}(A / \Sigma_h) \subseteq \mathcal{L}(A \backslash \Sigma_h)$.

Example 3.3: Let us consider the automaton A_a of Figure 1 (a) with $\Sigma_h = \{h\}$ and $\Sigma_l = \{\ell\}$. This automaton is not SNNI, because $\mathcal{L}(A \backslash \Sigma_h) = \varepsilon$ whereas $\mathcal{L}(A / \Sigma_h) = \ell$. The automaton A_b is SNNI.

As demonstrated by the following examples 3.4 and 3.5, a timed automaton A can be non-SNNI whereas its untimed underlying automaton is SNNI and A can be SNNI whereas its untimed underlying automaton is not.

Example 3.4: Let us consider the timed automaton A_g of Figure 2 (a), with $\Sigma_h = \{h\}$ and $\Sigma_l = \{\ell\}$. It is not SNNI since $(2.5, \ell)$ is accepted by A_g / Σ_h but not by $A_g \backslash \Sigma_h$. Its untimed underlying automaton A_h is SNNI since $\mathcal{L}(A_h \backslash \Sigma_h) = \{\ell\} = \mathcal{L}(A_h / \Sigma_h)$.

Example 3.5: Let us consider the timed automaton A_j of Figure 3 (a), with $\Sigma_h = \{h\}$ at $\Sigma_l = \{\ell_1, \ell_2\}$. It is SNNI, since $\mathcal{L}(A_j \backslash \Sigma_h) = \mathcal{L}(A_j / \Sigma_h)$. Its untimed underlying automaton A_k is not SNNI since $\ell_1 \cdot \ell 2$ is accepted by A_k / Σ_h but not by $A_k \backslash \Sigma_h$.

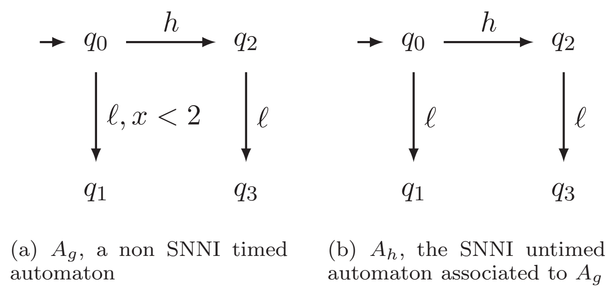

q_0
h
q_2
$\ell, x < 2$
ℓ
q_1
q_3
(a) A_g, a non SNNI timed automaton
q_0
h
q_2
ℓ
ℓ
q_1
q_3
(b) A_h, the SNNI untimed automaton associated to A_g

Figure 2. A non-SNNI timed automaton and its untimed underlying automaton which is SNNI.

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, "h"] \arrow[d, "{\ell_1, x > 2}"'] & q_3 \arrow[d, "{\ell_1, x > 2}"] \\
& q_1 \arrow[d, "\ell_1"'] & q_4 \arrow[d, bend right, "{\ell_2, x < 2}"'] \arrow[d, bend left, "\ell_1"] \\
& q_2 & q_5
\end{tikzcd}$$

(a) A_j, a SNNI timed automaton

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, "h"] \arrow[d, "\ell_1"'] & q_3 \arrow[d, "\ell_1"] \\
& q_1 \arrow[d, "\ell_1"'] & q_4 \arrow[d, bend right, "\ell_2"'] \arrow[d, bend left, "\ell_1"] \\
& q_2 & q_5
\end{tikzcd}$$

(b) A_k, the non SNNI untimed automaton associated to A_j

Figure 3. An SNNI timed automaton and its untimed underlying automaton which is non-SNNI.

Example 3.6 (SNNI): Figure 4 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 $1 \leq \delta < 2$ as 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 $A(2)/\Sigma_h \approx_{\mathcal{L}} A(2)\backslash\Sigma_h$.

Finally, since SNNI is based on language equivalence, we have the following lemma:

Lemma 3.7: *If $A' \approx_{\mathcal{L}} A$, then A is SNNI $\Leftrightarrow$ A' is SNNI.*

Proof: First, $\mathcal{L}(A/\Sigma_h) = \mathbf{proj}_{\Sigma_l}(\mathcal{L}(A)) = \mathbf{proj}_{\Sigma_l}(\mathcal{L}(A')) = \mathcal{L}(A'/\Sigma_h)$. Second, $\mathcal{L}(A\backslash\Sigma_h) = \mathcal{L}(A) \cap T\Sigma_l^* = \mathcal{L}(A') \cap T\Sigma_l^* = \mathcal{L}(A'\backslash\Sigma_h)$. □

3.2 *Cosimulation strong non-deterministic non-interference (CSNNI)*

The CSNNI property has been introduced in Gardey et al. (2005), and is based on *cosimulation*.

Definition 3.8: A timed automaton A is *CSNNI* iff $A\backslash\Sigma_h \approx_{\mathcal{CW}} A/\Sigma_h$.

Since $A/\Sigma_h \sqsubseteq_{\mathcal{W}} A\backslash\Sigma_h$, we can give a simple characterisation of CSNNI:

Proposition 3.9: *A timed automaton A is CSNNI iff $A\backslash\Sigma_h \sqsubseteq_{\mathcal{W}} A/\Sigma_h$.*

By restricting the class of TA considered, we obtain the following result.

$$\begin{tikzcd}
{} \arrow[r] & 0 \arrow[r, "{h, x \geq k}"] \arrow[d, "{l, x \geq 2}"'] & 2 \arrow[d, "l"] \\
& 1 & 3
\end{tikzcd}$$

Figure 4. Automaton $A(k)$.

Example 3.10: Let us consider the automaton A_c of Figure 5 (a) with $\Sigma_h = \{h\}$ and $\Sigma_l = \{\ell_1, \ell_2, \ell_3\}$. A_c is SNNI but is not CSNNI, because no state of $A_c\backslash\Sigma_l$ can simulate the state q_6. The automaton A_d of Figure 5(a) is CSNNI. The state q_1 of $A_d\backslash\Sigma_l$ simulates the states q_5 and q_6.

We complete this sub-section by comparing SNNI and CSNNI. Given two timed automata A_1 and A_2, $A_1 \sqsubseteq_{\mathcal{W}} A_2$ implies that $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$. CSNNI is thus stronger than SNNI as for each timed automaton A, $A\backslash\Sigma_h \sqsubseteq_{\mathcal{W}} A/\Sigma_h$ implies that $\mathcal{L}(A/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$.

The converse holds when $A\backslash\Sigma_h$ is deterministic.

Lemma 3.11: *If $A\backslash\Sigma_h$ is deterministic, then A is SNNI implies that A is CSNNI.*

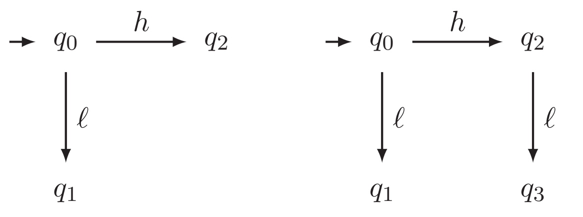
$$\begin{tikzcd}
{} \arrow[r] & & q_0 \arrow[rrr, "h"] \arrow[dl, "\ell_1"'] \arrow[dr, "\ell_1"] & & & q_5 \arrow[d, "\ell_1"] & \\
& q_1 \arrow[d, "\ell_2"] & & q_2 \arrow[d, "\ell_3"] & & q_6 \arrow[dl, "\ell_2"'] \arrow[dr, "\ell_3"] & \\
& q_3 & & q_4 & q_7 & & q_8
\end{tikzcd}$$

(a) A_c, a SNNI but not CSNNI automaton

$$\begin{tikzcd}
{} \arrow[r] & & q_0 \arrow[rrr, "h"] \arrow[d, "\ell_1"] & & & q_4 \arrow[dl, "\ell_1"'] \arrow[dr, "\ell_1"] & \\
& & q_1 \arrow[dl, "\ell_2"'] \arrow[dr, "\ell_3"] & & q_5 \arrow[d, "\ell_2"] & & q_6 \arrow[d, "\ell_3"] \\
& q_2 & & q_3 & q_7 & & q_8
\end{tikzcd}$$

(b) A_d, a CSNNI automaton

Figure 5. CSNNI is stronger than SNNI.



(a) A_e, a CSNNI but not BSNNI automaton (b) A_f, a BSNNI automaton

Figure 6. BSNNI is stronger than CSNNI.

Proof: As emphasised before, given two timed automata A_1 and A_2, $A_1 \sqsubseteq_{\mathcal{W}} A_2$ implies that $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$. If A_1 is deterministic, then $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ implies that $A_1 \sqsubseteq_{\mathcal{W}} A_2$. To obtain the result, it suffices to take $A_1 = A \backslash \Sigma_h$ and $A_2 = A/\Sigma_h$. □

3.3 *Bisimulation strong non-deterministic non-interference (BSNNI)*

The BSNNI property has been introduced in Focardi and Gorrieri (2001) and is based on bisimulation.

Definition 3.12: A timed automaton A is BSNNI iff $A\backslash\Sigma_h \approx_{\mathcal{W}} A/\Sigma_h$.

The automation A_f of Figure 6(b) is BSNNI. Bisimulation is stronger than cosimulation and we have for a timed automaton A, if A is BSNNI, then A is CSNNI (and thus A is SNNI).

As the following example demonstrates, there exists an automaton which is CSNNI and not BSNNI.

Example 3.13: Let us consider the automaton A_e of Figure 6(a) with $\Sigma_h = \{h\}$ et $\Sigma_l = \{\ell\}$. This automaton is deterministic and SNNI, and therefore by lemma 3.11, it is CSNNI. However, it is not BSNNI, since the state q_2 of A_e/Σ_h has no bisimilar state in $A_e\backslash\Sigma_h$.

4. Verification of non-interference properties for timed automata

In this section, we settle the complexity of non-interference verification problems for TA.

4.1 *SNNI verification*

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

For TA, this problem has been proved to be *undecidable* in Gardey et al. (2005) and the proof is based on the fact that language containment for TA is undecidable (Alur & Dill, 1994). However, if we consider the sub-class of timed automata A such that $A\backslash\Sigma_h$ is *deterministic*, then the problem becomes decidable. In the sequel, we called dTA the class of timed automata A such that $A\backslash\Sigma_h$ is deterministic. Remark that, since finite automata are a special case of timed automata (TA with no clock), dTA also contains finite automata.

Theorem 4.1: *The SNNI-VP is PSPACE-complete for* dTA.

Proof: Let A_1 and A_2 be two TA. Checking whether $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ with A_1 a deterministic TA is PSPACE-complete (Alur & Dill, 1994). Checking $\mathcal{L}(A/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$ can thus be done in PSPACE if $A\backslash\Sigma_h$ is deterministic. Using Proposition 3.2, it follows that SNNI-VP is PSPACE-easy for dTA.

For PSPACE-hardness, we reduce the language inclusion problem $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$, with A_1 a deterministic TA, to the SNNI-VP. Let $A_1 = (Q_1, q_{01}, X_1, \Sigma, E_1, Inv_1)$ be a deterministic TA and $A_2 = (Q_2, q_{02}, X_2, \Sigma, E_2, Inv_2)$ a TA.[2] We let $h \notin \Sigma$ be a fresh letter, $x \notin X_1 \cup X_2$ be a fresh clock, and define $A_{12} = (\{q_{12}^0\} \cup Q_1 \cup Q_2, q_{12}^0, X_1 \cup X_2 \cup \{x\}, \Sigma^\varepsilon \cup \{h\}, E_{12}, Inv_{12})$ be the timed automaton defined (as shown in Figure 7) as follows:

- the transition relation E_{12} contains $E_1 \cup E_2$ and the additional transitions $(q_{12}^0, true, h, \varnothing, q_{02})$ and $(q_{12}^0, true, \varepsilon, \varnothing, q_{01})$;
- $Inv_{12}(q) = Inv_i(q)$ if $q \in Q_i, i \in \{1, 2\}$, and $Inv_{12}(q_{12}^0) = [x \leq 0]$.

We let $\Sigma_l = \Sigma$ and $\Sigma_h = \{h\}$. We prove that A_{12} is SNNI iff $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$. This is easily established as

$$\begin{aligned}
&A_{12} \text{ is SNNI}\\
&\text{iff } \mathcal{L}(A_{12}/\Sigma_h) \subseteq \mathcal{L}(A_{12}\ \Sigma_h) \quad \text{[Proposition 3.2]}\\
&\text{iff } \mathcal{L}(A_1) \cup \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)\\
&\text{iff } \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1).
\end{aligned}$$

Thus, the SNNI-VP is PSPACE-complete for dTA. □

For non-deterministic finite automata A_1 and A_2, checking language inclusion $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ is PSPACE-complete (Stockmeyer & Meyer, 1973). Then, using the same proof with A_1 being a non-deterministic finite automaton as follows.

Corollary 4.2: *The SNNI-VP is PSPACE-complete for non-deterministic finite automata.*

Moreover, when A_2 is a deterministic finite automaton, language containment can be checked in PTIME and thus we have the following corollary.

Corollary 4.3: *For finite automata belonging to* dTA, *the SNNI-VP is PTIME.*

Table 1 summarises the results on the complexity of the SNNI-VP.

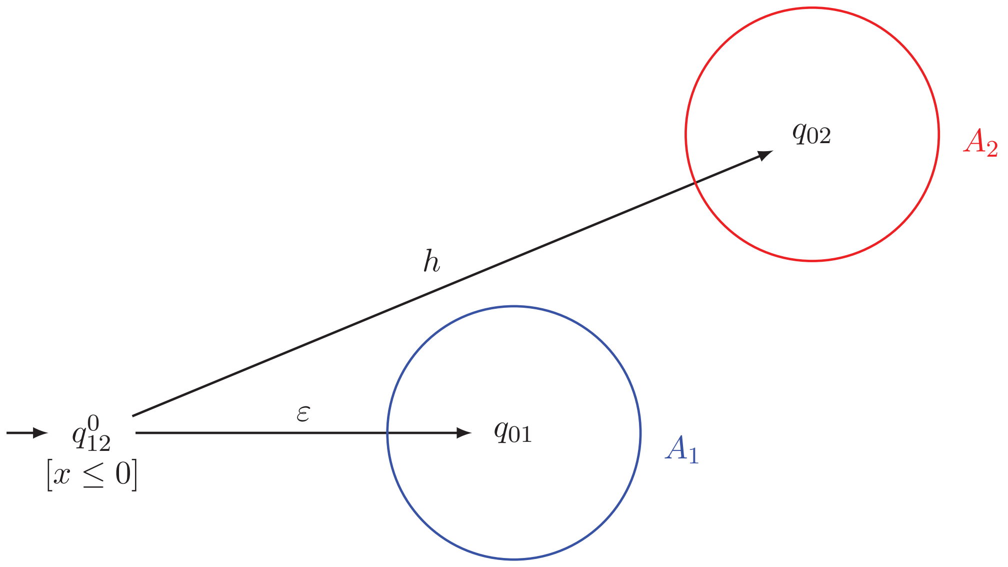


Figure 7. The timed automaton A_{12}.

Table 1. Complexity if SNNI-VP.

	Timed automata	Finite automata
$A\backslash\Sigma_h$ is *deterministic* (dTA)	PSPACE-complete (Thm 4.1)	PTIME (Cor. 4.3)
General Case	Undecidable (Gardey et al., 2005)	PSPACE-complete (Cor. 4.2)

4.2 *Verification of CSNNI and BSNNI properties*

BSNNI-VP and CSNNI-VP are decidable for TA (Gardey et al., 2005) since simulation and bisimulation are decidable. For finite automata, the complexity of BSNNI-VP and CSNNI-VP is known to be PTIME (Cassez et al., 2007). We settle here the complexity of those problems for tTA.

Theorem 4.4: *The CSNNI-VP and BSNNI-VP are EXPTIME-complete for TA.*

Proof: Strong timed bisimilarity and simulation pre-order are both EXPTIME-complete for TA. The EXPTIME-hardness is established in Laroussinie and Schnoebelen (2000), where it is shown that any relation between simulation pre-order and bisimilarity is EXPTIME-hard for TA.

The EXPTIME-easiness for strong timed bisimulation was established in Čerāns (1992) and for simulation pre-order in Tasiran, Alur, Kurshan, and Brayton (1996).

To establish EXPTIME-completeness for CSNNI-VP and BSNNI-VP, we show that these problems are equivalent to their counterparts for TA.

To do this, we use the automata A_1, A_2, and A_{12} already defined in the proof of Theorem 4.1.

We show that A_1 simulates A_2 iff A_{12} is CSNNI.

Assume that A_1 simulates A_2. There exists a relation $\mathcal{R}$, s.t. (1) $(q_{01}, \vec{0}_{X_1})\mathcal{R}(q_{02}, \vec{0}_{X_2})$ and (2) for each state $(s_2, \vec{x_2})$, there exists $(s_1, \vec{x_1})$ s.t. $(s_1, \vec{x_1})\mathcal{R}(s_2, \vec{x_2})$, and whenever $(s_2, \vec{x_2}) \xrightarrow{a} (s_2', \vec{x_2}')$ for $a \in \Sigma \cup \mathbb{R}_+$, then $(s_1, \vec{x_1}) \xrightarrow{a} (s_1', \vec{x_1}')$ and $(s_1', \vec{x_1}')\mathcal{R}(s_2', \vec{x_2}')$.

We define a relation $\mathcal{R}'$ for each $(\ell, \vec{x_1}\vec{x_2}x)$ of A_{12}/Σ_h to a state $(\ell', \vec{x_1}'\vec{x_2}'x')$ of $A_{12}\backslash\Sigma_h$ as follows:

- if $\ell = q_{12}^0$ then $(\ell, \vec{x_1}\vec{x_2}x)\mathcal{R}'(\ell, \vec{x_1}'\vec{x_2}'x')$;
- if $\ell \in Q_1$, then $(\ell, \vec{x_1}\vec{x_2}x)\mathcal{R}'(\ell, \vec{x_1}\vec{x_2}'x')$;
- if $\ell \in Q_2$, then $(\ell, \vec{x_1}\vec{x_2}x)\mathcal{R}'(\ell', \vec{x_1}'\vec{x_2}'x')$ iff $(\ell, \vec{x_2})\mathcal{R}(\ell', \vec{x_1})$;

$\mathcal{R}'$ is a simulation of A_{12}/Σ_h by $A_{12}\backslash\Sigma_h$:

- the initial states of the two TA are in relation;
- assume $(s, \vec{x_1}\vec{x_2}x) \xrightarrow{a}_{A_{12}/\Sigma_h} (s', \vec{x_1}'\vec{x_2}'x')$; If $s \in \{q_{12}^0\} \cup Q_1$, then clearly it is simulated by the same state in $A_{12}\backslash\Sigma_h$. Otherwise, if $s \in Q_2$, then there exists a state $(\ell', \vec{x_1}\vec{x_2}'x')$ in $A_{12}\backslash\Sigma_h$ s.t. $(s, \vec{x_1}\vec{x_2}x)\mathcal{R}'(s', \vec{x_1}'\vec{x_2}'x')$: by definition of $\mathcal{R}'$, we can take any $(s', \vec{x_1}'\vec{x_2}'x')$ with $(s, \vec{x_2})\mathcal{R}(s', \vec{x_1}')$. It is easy to see that because A_1 can simulate A_2, from there on, $\mathcal{R}'$ is indeed a simulation relation. Thus, A_{12}/Σ_h and $A_{12}\backslash\Sigma_h$ are cosimilar by Proposition 3.9.

Now assume conversely that there is a simulation $\mathcal{R}'$ of A_{12}/Σ_h by $A_{12}\backslash\Sigma_h$. We can define a simulation relation of A_2 by A_1 as follows: each state $(s, \vec{x_1}\vec{x_2}x)$ with $s \in Q_2$ of A_{12}/Σ_h is simulated by a state $(s', \vec{x_1}'\vec{x_2}'x')$ with $s' \in Q_1$. We then define $\mathcal{R}$ by $(s, \vec{x_2})\mathcal{R}(s', \vec{x_1}')$. Again, it is easy to see that $\mathcal{R}$ is a simulation relation.

It follows that CSNNI is EXPTIME-complete.

Table 2. Results for CSNNI-VP and BSNNI-VP.

	Timed automata	Finite automata
CSNNI-VP	EXPTIME-C (Theorem 4.4)	PTIME (Cassez et al., 2007)
BSNNI-VP	EXPTIME-C (Theorem 4.4)	PTIME (Cassez et al., 2007)

Now assume that A_1 and A_2 are bisimilar. We can define the relation $\mathcal{R}'$ exactly as above and this time it is a weak bisimulation between $A_{12}\backslash\Sigma_h$ and A_{12}/Σ_h.

If A_{12} is BSNNI, the bisimulation relation $\mathcal{R}'$ between $A_{12}\backslash\Sigma_h$ and A_{12}/Σ_h induces a bisimulation relation $\mathcal{R}$ between A_1 and A_2: it suffices to build $\mathcal{R}$ as the restriction of $\mathcal{R}'$ between states with a discrete component in Q_1 and a discrete component in Q_2.

As checking bisimulation between TA is also EXPTIME-complete, the EXPTIME-completeness of BSNNI-VP for TA follows. □

Table 2 summarises the results on the verification of the CSNNI and BSNNI properties.

5. The SNNI control problem

The previous non-interference verification problem consists of *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 behaviour of users to ensure a given 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. Note that, contrary to Cassez et al. (2007), we release the constraint $\Sigma_c = \Sigma_h$. The motivations for this work are many fold. 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.

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:

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

The SNNI-*controller synthesis problem* (SNNI-CSP) asks to compute a witness when the answer to the SNNI-CP is 'yes'.

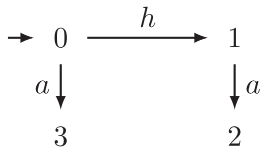

$$\begin{tikzcd}
\arrow[r] & 0 \arrow[r, "h"] \arrow[d, "a"'] & 1 \arrow[d, "a"] \\
& 3 & 2
\end{tikzcd}$$

Figure 8. Automaton D.

While the control properties for SNNI seem quite close to the corresponding classical problems of control with partial observability, we show in Section 5.4 that, surprisingly, the link between the two is not trivial.

5.1 *Preliminary remarks*

First, we motivate our definition of controllers that 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.

Example 5.1: Let us consider the automaton D of Figure 8 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.

5.2 *SNNI-VP versus SNNI-CP*

SNNI-CP is harder than SNNI-VP since SNNI-VP reduces to SNNI-CP by taking $\Sigma_c = \varnothing$. Note that this is not true if we restrict to the sub-class of control where $\Sigma_c = \Sigma_h$. Indeed, in this case, SNNI-CP is always true (and then decidable) since the controller which forbids all controllable transitions make the system SNNI.

We then have the following theorem.

Theorem 5.2: *For general TA, SNNI-CP and SNNI-CSP are undecidable.*

Proof: SNNI-CP obviously reduces to SNNI-CSP. SNNI-VP reduces to SNNI-CP by taking $\Sigma_c = \varnothing$. SNNI-VP is undecidable for non-deterministic TA. □

We will now show that SNNI-CP reduces to the SNNI-VP for finite automata.

Proposition 5.3: *Let A be a finite automaton. There exists a controller C such that $C(A)$ is SNNI iff $A\backslash\Sigma_c$ is SNNI.*

Proof: Since time is not taken into account in untimed automata, we can have $C(\rho) = \varnothing$ for finite automaton (for general timed automaton, this would mean that we block the time). The proof of the theorem consists of 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 $C_\forall(\rho) = \varnothing$. 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 the following:

- 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 $e \in \Sigma_h$, $\mathcal{L}((D\backslash\{e\})/\Sigma_h) \subseteq \mathcal{L}(D/\Sigma_h) = \mathcal{L}(D\backslash\Sigma_h) = \mathcal{L}((D\backslash\{e\})\backslash\Sigma_h)$.

Therefore, if D is SNNI, $D\backslash L$ is SNNI, $\forall L \subseteq \Sigma$. 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. □

Hence the following result.

Theorem 5.4: *For finite automata, the SNNI-CP is PSPACE-complete.*

Proof: Let A be a finite automaton. By Proposition 5.3, it is 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 $\Sigma_c = \varnothing$. □

Moreover, since the SNNI-CP reduces to the SNNI-VP for finite automata, and from Corollary 4.3, we have the following result:

Corollary 5.5: *For finite automata belonging to dTA, the SNNI-CP is PTIME.*

Proposition 5.3 does not hold for general TA as the following example demonstrates.

Example 5.6: Figure 9 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

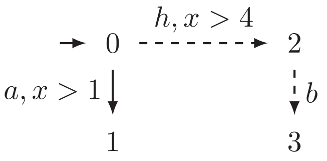


Figure 9. The Automaton H.

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.

5.3 *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 5.7: *For dTA, the SNNI-CP is EXPTIME-hard.*

Proof: The safety control problem for TA is EXPTIME-hard (Henzinger & Kopke, 1997). In the proof of this theorem, Henzinger and Kopke use TA 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 Henzinger and Kopke (1997) to TA which are deterministic with respect to Σ_c actions and non-deterministic with respect to 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: $(bad, \mathsf{true}, h, \varnothing, q_h)$ and $(q_h, \mathsf{true}, l, \varnothing, q_l)$, 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 cannot be fired. Thus, if there is controller 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 $2^{\Sigma_c\cup\{\lambda\}}\setminus\varnothing$) 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 that $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$ 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 $A \setminus \Sigma_h$ (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 as $last_2(w)$ 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 5.8: : 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. $\xrightarrow{p}$ 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 $\xrightarrow{p}$.

Lemma 5.9: *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 $\mathbf{proj}_{\Sigma_l}(w) \notin \mathcal{L}(A\backslash\Sigma_h)$.*

The proof follows easily from Fact 5.8. 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[3] in A_2 whose trace is $\mathbf{proj}_{\Sigma_l}(trace(\rho))$. 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 5.10: *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, $Reach(C^\otimes(A_p)) \cap Bad^\otimes = \varnothing$.*

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 $Reach(C^\otimes(A_p)) \cap Bad^\otimes \neq \varnothing$. By definition, there is a run ρ' in $Runs(C^\otimes(A_p))$. such that

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

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 $1 \le i \le n$ (and $q_0^2 \notin Bad$). Let $\rho = \rho'_{|1}$ and $w = \mathbf{proj}_{\Sigma_l}(trace(\rho')) = \mathbf{proj}_{\Sigma_l}(trace(\rho))$. 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 5.9. 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 $Reach(C^\otimes(A_p)) \cap Bad^\otimes = \varnothing$. □

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 emphasised before, this does not necessarily ensure that $C(A)$ is SNNI.

Lemma 5.11: *Let $C^\otimes$ be a controller for A_p s.t. $Reach(C^\otimes(A_p)) \cap Bad^\otimes = \varnothing$. Let $C(\rho) = C^\otimes(\rho')$ if $\rho'_{|1} = \rho$. C is well-defined and $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$.*

Proof: Let $\rho = (q_0, \vec{0}) \xrightarrow{e_1} (q_1, v_1) \xrightarrow{e_2} \cdots \xrightarrow{e_n} (q_n, v_n)$ be a run of A. Since A_2 is deterministic and complete, there is exactly one run $\rho' = ((q_0, q_0), (\vec{0}, \vec{0})) \xrightarrow{e_1} ((q_1, q_1'), (v_1, v_1')) \xrightarrow{e_2} \cdots \xrightarrow{e_n} ((q_n, q_n'), (v_n, v_n'))$ in A_p s.t. $\rho'_{|1} = \rho$. Therefore, 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 $Runs(C(A)) \subseteq Runs(A)$ s.t. $\mathbf{proj}_{\Sigma_l}(trace(\rho)) = w$, there is a unique run $\rho \in Runs(A_p)$ s.t. $\rho'_{|1} = \rho$ and $trace(\rho') = w$. First, by Lemma 5.9, $last(\rho') \in Bad^\otimes$. Second, this run ρ' is in $Runs(C^\otimes(A_p))$ because of the definition of C. Hence, $Reach(C^\otimes(A_p)) \cap Bad^\otimes \neq \varnothing$ 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 that the safety game $(A_p, Bad^{\otimes})$ can be won and $C^{\otimes}$ is the most permissive controller. Let C be the controller obtained using Lemma 5.11. 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 5.12: Consider the timed automaton K on the left of Figure 10 with $\Sigma_h = \{h\}$ and $\Sigma_c = \{a\}$.

We can compute the complete version of $K \setminus \Sigma_h$, which we call K_2. This is the middle automaton in Figure 10.

Then, we compute the product $K \times_{\Sigma_l} K_2$ of K and K_2 with synchronisation only on the Σ_l actions. This corresponds to the rightmost automaton in Figure 10.

Notice that there is only one location with a second component equal to Bad in this product. So $Bad^{\otimes} = \{(3, Bad)\}$. By a classical safety control algorithm, we can then compute the most permissive controller $C^{\otimes}$ to avoid $Bad^{\otimes}$, which consists in cutting the a between locations 0 and 1 at all times, because a is the only controllable action.

By projecting $C^{\otimes}$ on the first component as defined in Lemma 5.11, we obtain C: $C(K)$ as given by the sub-automaton of K with the plain arrows. $C(K)$ is obviously not SNNI.

Example 5.13: For the example of $A(1)$ in Figure 4, if we compute C as described above, 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 strengthened).

The example of Figure 10 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 5.14: *Consider the controller C as defined in Lemma 5.11. If $C(A)\backslash\Sigma_h \approx_{\mathcal{L}} A\backslash\Sigma_h$, then $C(A)$ is SNNI.*

Proof: If $C(A)\backslash\Sigma_h \approx_{\mathcal{L}} A\backslash\Sigma_h$, 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, therefore, $C(A)$ 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 $A^i_p = A^i \times_{\Sigma_l} A^i_2$ and $C^{\otimes}_{i+1}$ be the most permissive controller for the safety game $(A^i_p, Bad^{\otimes}_i)$ ($\perp$ if no such controller exists). We use the notation $Bad^{\otimes}_i$ because this set depends on A^i_2. We define C^{i+1} using Lemma 5.11: $C^{i+1}(\rho) = C^{\otimes}_{i+1}(\rho')$ if $\rho'_{|1} = \rho$. Let $A^{i+1} = C^{i+1}(A^i)$.

By Lemma 5.14, if $C^{i+1}(A^i)\backslash\Sigma_h \approx_{\mathcal{L}} A^i\backslash\Sigma_h$, then $C^{i+1}(A^i)$ is SNNI. Therefore, this condition is a sufficient condition for the termination of the algorithm defined above.

Lemma 5.15: *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 sub-graph of the region graph of $C^1(A^0)$ for $i \geq 1$. By Lemma 5.11 (and the remark following it), $C^1(A^0)$ is a sub-graph 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 sub-graph of $RG(A^{k-1})$ for $k \geq 2$ and $RG(A^{k-1}\backslash\Sigma_h)$ is a sub-graph of $RG(A\backslash\Sigma_h)$. Using Lemma 5.11, we can compute $A^k = C^k(A^{k-1})$ and (1) $RG(A^k\backslash\Sigma_h)$ is a sub-graph of $A^{k-1}\backslash\Sigma_h$ and (2) the memory needed for $C^{\otimes}_k$ 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 sub-graph of $RG(A^{k-1})$.

The most permissive controller $C^{\otimes}_i$ will either disable at least one controllable transition of $A^{i-1}_p\backslash\Sigma_h$ or keep all the controllable transitions of $A^{i-1}_p\backslash\Sigma_h$. 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 $A^i\backslash\Sigma_h \approx_{\mathcal{L}} A^{i-1}\backslash\Sigma_h$ (Lemma 5.14) 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 5.16: *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 $\forall \rho \in Runs(A)$, $M(\rho) \subseteq C^i(\rho)$.*

Proof: The proof is by induction:

- For $i = 0$, it holds trivially.
- Assume that the lemma holds for indices up until i. Thus, we have $Runs(M(A)) \subseteq Runs(A^i)$. Therefore, we can define M over A^i and $M(A^i)$ is SNNI. By Lemma 5.10, $M^{\otimes}$ is a controller for the safety game $(A^i_p, Bad^{\otimes}_i)$, therefore ,$M^{\otimes}(\rho') \subseteq C^{\otimes}_{i+1}(\rho')$ because $C^{\otimes}_{i+1}$ is the most permissive controller. This implies that $M(\rho) \subseteq C^{i+1}(\rho)$ by definition of C^{i+1}.

□

Using Lemma 5.15, the sequence C^i converges to a fix point. Let C^* denote this fix point.

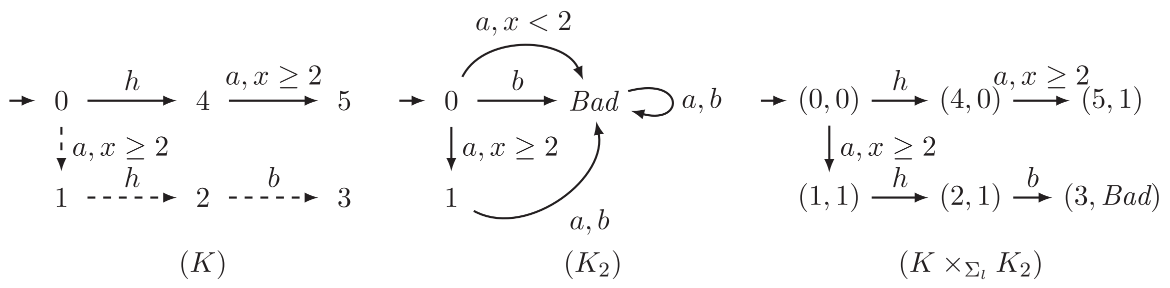


Figure 10. The Automata K, K_2, and $K \times_{\Sigma_l} K_2$.

Lemma 5.17: *C^* is the most permissive controller for the SNNI-CSP.*

Proof: Either $C^* = \perp$ and there is no way of enforcing SNNI (Lemma 5.10), or $C^* \neq \perp$ is such that $\mathcal{L}(C^*(A)/\Sigma_h) = \mathcal{L}(C^*(A)\backslash\Sigma_h)$ by Lemma 5.11. 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 5.16), the result follows. □

Lemma 5.15 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 5.18: *For a dTA A, C^* can be computed in $O(2^{4.|A|})$.*

Proof: As the proof of Lemma 5.15 shows, the region graph of A^i is a sub-graph 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 $Bad^{\otimes}_i$ can be done in linear time in the size of the region graph of A^i_p. As $RG(A^i)$ is a sub-graph of $RG(A^1)$, $RG(A^i_p)$ is a sub-graph of $RG(A^1_p)$. Therefore, we have to solve at most $|RG(A)|$ safety games of sizes at most $|RG(A^1_p)|$. As A^1 is a sub-graph of $A^0_p = A^0 \times_{\Sigma_l} A^0_2$, $|RG(A^1)| \leq |RG(A)|^2$. And, as $A^1_p = A^1 \times_{\Sigma_l} A^1_2$, $|RG(A^1_p)| \leq |RG(A)|^3$. Therefore, C^* can be computed in $O(|RG(A)|.|RG(A^1_p)|) = O(|RG(A)|^4) = O(2^{4.|A|})$. □

Theorem 5.19: *For dTA, the SNNI-CP and SNNI-CSP are EXPTIME-complete.*

For the special case of finite automata, we even have the following:

Lemma 5.20: *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 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. $\mathbf{proj}_{\Sigma_l}(u.l.m) \in \mathcal{L}(C^1(A)/\Sigma_h)$ but $\mathbf{proj}_{\Sigma_l}(u.l.m) \notin \mathcal{L}(C^1(A)\backslash\Sigma_h)$. But by definition of C^1, $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A\backslash\Sigma_h)$ (Lemma 5.11), and thus $\mathbf{proj}_{\Sigma_l}(u.l.m) \in \mathcal{L}(A\backslash\Sigma_h)$. As $u.l \in \Sigma_l^*$, $\mathbf{proj}_{\Sigma_l}(u.l.m) = u.l.\mathbf{proj}_{\Sigma_l}(m)$ and $\mathbf{proj}_{\Sigma_l}(m) \in \Sigma_u^*$. Since $u.l \in \mathcal{L}(C^1(A)\backslash\Sigma_h)$ and $\mathbf{proj}_{\Sigma_l}(m) \in \Sigma_u^*$, we have $u.l.\mathbf{proj}_{\Sigma_l}(m) \in \mathcal{L}(C^1(A)\backslash\Sigma_h)$ which is a contradiction. Thus, $\mathcal{L}(C^2(A)\backslash\Sigma_h) = \mathcal{L}(C^1(A)\backslash\Sigma_h)$ which is our stopping condition by Lemma 5.14 and thus $C^* = C^2$. □

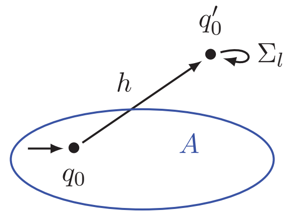


Figure 11. Automaton B.

It follows that

Theorem 5.21: *For a finite automaton A in dTA (i.e. such that $A\backslash\Sigma_h$ is deterministic), the SNNI-CSP is PSPACE-complete.*

As untimed automata can always be determinised, we can extend our algorithm to untimed automata when $A\backslash\Sigma_h$ non-deterministic. It suffices to determinise $A^i_2, i = 1, 2$.

Theorem 5.22: *For a finite automaton A such that $A\backslash\Sigma_h$ is non-deterministic, the SNNI-CSP can be solved in EXPTIME.*

Proposition 5.23: *There is a family of finite automata $(A_i)_{i\geq 0}$ such that (1) there is a most permissive controller D^*_i s.t. $D^*_i(A_i)$ is SNNI and (2) 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 11. 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 emphasised in Lemma 5.11. It needs finite memory on the remaining part, i.e. on B. The controller D on B gives

Table 3. Summary of the results for SNNI-CP and SNNI-CSP.

	A finite automaton	
	$A \backslash \Sigma_h$ Non-det.	$A \backslash \Sigma_h$ det.
SNNI-CP	PSPACE-C (Theorem 5.4)	PTIME (Corollary 5.5)
SNNI-CSP	EXPTIME (Theorem 5.22)	PSPACE-C (Theorem 5.21)
	A timed automaton	
	$A \backslash \Sigma_h$ Non-det.	$A \backslash \Sigma_h$ Det.
SNNI-CP	Undecidable (Theorem 5.2)	EXPTIME-C (Theorem 5.19)
SNNI-CSP	Undecidable (Theorem 5.2)	EXPTIME-C (Theorem 5.19)

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 finite automata, 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 (with respect to to A_i). This proves the EXPTIME lower bound. □

In this section, 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 determinised 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 determinised. We have provided an optimal algorithm to solve the SNNI-CP and SNNI-CSP in this case (although we have not proved a completeness result).

The summary of the results is given in Table 3.

5.4 *SNNI-CP versus control with partial observability*

While the SNNI-VP is easily reducible to a classical language inclusion problem, the situation is more subtle for the control problem: in this section, we emphasise the distinction between classical control with partial observability and SNNI control. First, recall that the SNNI controllability property is as follows:

$$\text{Is there a controller } C \text{ s.t. } \mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(C(A) \setminus \Sigma_h)?$$

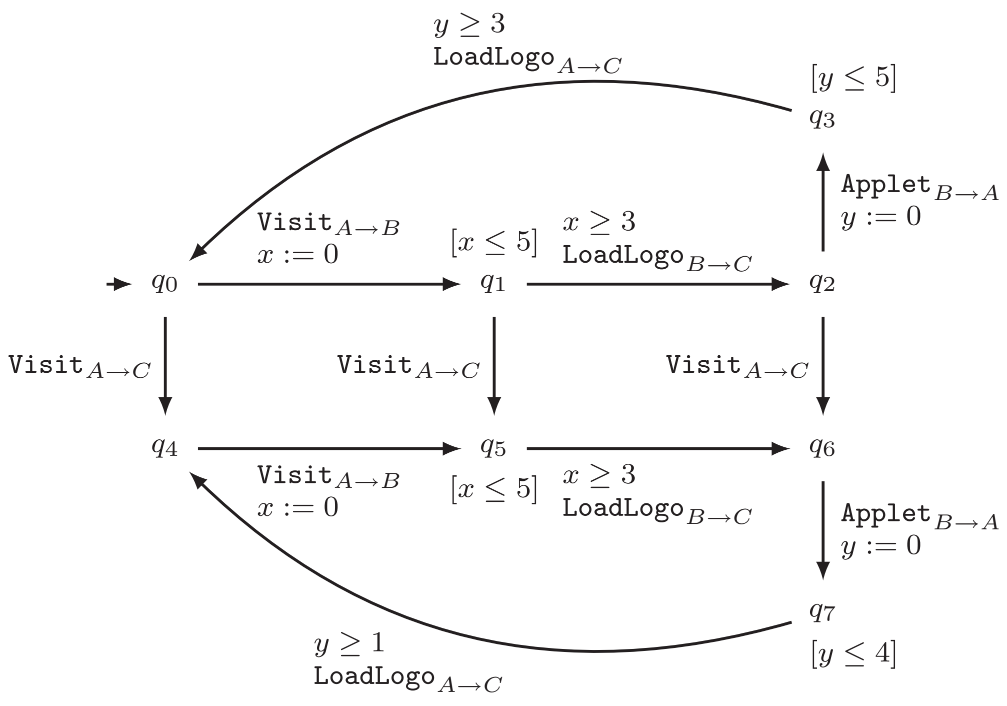
In contrast, in our setting, the corresponding property of controllability with partial observability (CPO) would be, for a set of unobservable (and uncontrollable) events Σ_{no} and a specification given as an automaton B:

$$\text{Is there a controller } C \text{ s.t. } \mathcal{L}(C(A)/\Sigma_{\text{no}}) \subseteq \mathcal{L}(B)?$$

We see that for the SNNI property, the controller is also modifying the target language, which is not trivial to account for in the setting of CPO.

Now, even if we can find a clever polynomial encoding of the SNNI-CP and SNNI-CSP into the framework of CPO, this would be interesting mostly for the finite automaton case. In this case, partial observability basically reduces to full observability (Kupferman & Vardi, 1997; Lamouchi & Thistle, 2000; Lin & Wonham, 1988), with the same EXPTIME complexity we obtain for the SNNI-CSP (recall that the SNNI-CP reduces to the simpler SNNI-VP for finite automata).

For TA, we have proved that the SNNI-CP and SNNI-CSP problems are undecidable in the general case. And so are the corresponding CPO problems (D'Souza & Madhusudan, 2002). If we restrict to the class dTA that we have exhibited, however, we have proved that these SNNI control problems are EXPTIME-complete. If we consider the most favourable case, in which we could polynomially encode the SNNI control problems for dTA in the framework of CPO with a deterministic target language, the complexity of the resulting algorithm would be 2EXPTIME as the CPO problem with fixed resources and a deterministic external specification is 2EXPTIME-complete (Bouyer et al., 2003). Our custom algorithm would, therefore, be a very important improvement compared to this hypothetical approach.



$y \geq 3$
$\texttt{LoadLogo}_{A\to C}$
$[y \leq 5]$
q_3
$\texttt{Applet}_{B\to A}$
$y := 0$
$\texttt{Visit}_{A\to B}$
$x := 0$
$[x \leq 5]$
$x \geq 3$
$\texttt{LoadLogo}_{B\to C}$
q_0
q_1
q_2
$\texttt{Visit}_{A\to C}$
$\texttt{Visit}_{A\to C}$
$\texttt{Visit}_{A\to C}$
q_4
q_5
q_6
$\texttt{Visit}_{A\to B}$
$x := 0$
$[x \leq 5]$
$x \geq 3$
$\texttt{LoadLogo}_{B\to C}$
$\texttt{Applet}_{B\to A}$
$y := 0$
q_7
$y \geq 1$
$\texttt{LoadLogo}_{A\to C}$
$[y \leq 4]$

Figure 12. The automaton W modelling the web privacy problem.

5.5 *Application to timing attacks on web privacy*

We illustrate the SNNI-CP for the class of timing attacks described in Felten and Schneider (2000), Bortz and Boneh (2007), and Kotcher et al. (2013), which can compromise the privacy of the Web browsing histories of users. The attacks allow a malicious website to determine whether or not the user has recently visited some other unrelated webpage. For example, an insurance company site could determine whether the user has recently visited websites relating to a particular medical condition; or an employer's website could determine whether an employee visiting it has recently visited the sites of various political organisations.

We use the simple example proposed in Felten and Schneider (2000) in which a malicious page can determine this information by measuring the time the user's browser requires to perform certain operations. Suppose that Alice is surfing on the Web, and she visits Bob's website. Bob wants to find out whether Alice has visited Charlie's website. First, Bob looks at Charlie's site, and picks a file that any visitor to the site will have seen. Bob picks the file `logo.jpg` containing Charlie's corporate logo. Bob is going to determine whether the logo file is in Alice's webbrowser cache. If the file is in her cache, then she must have visited Charlie's website recently. Bob writes a Java applet that implements his attack, and embeds it in his home page. When Alice views the attack page, the Java applet is automatically downloaded and run in Alice's browser. The applet measures the time required to access `logo.jpg` on Alice's machine, and reports this time back to Bob. According to this time, Bob may conclude that Alice has been to Charlie's site recently.

Figure 12 gives the timed automaton W modelling this example with high-level action $\Sigma_h = \{\texttt{Visit}_{A\to C}\}$ and all other actions are low-level actions (i.e. in Σ_l). Notice that W is not SNNI since the timed word $\rho = \ldots(\delta, \texttt{Applet}_{B\to A})(1, \texttt{LoadLogo}_{A\to C})\ldots$, for any delay $\delta \geq 0$, is in $\mathcal{L}(W/\Sigma_h)$ and not in $\mathcal{L}(W\backslash\Sigma_h)$.

Assume $\Sigma_c = \{\texttt{LoadLogo}_{A\to C}\}$. As we have seen before, we can compute K_2, a copy of $W\backslash\Sigma_h$ completed with a Bad location. Then, we build the product $W \times_{\Sigma_l} K_2$, synchronised on Σ_l, of W and K_2. Both K_2 and W are too big for us to depict here while maintaining clarity, so we give in Figure 13 only the crucial part of this product where the interference may occur. The dotted arrows denote parts of the product that have been abstracted away.

We can see that, as expected, an interference may occur if the logo is loaded faster than three time units, which corresponds to the transition from (q_7, q_3) to Bad in Figure 13. Then, $\texttt{LoadLogo}_{A\to C}$ being the only controllable action, the solution to the safety control problem we obtain consists of cutting that transition to Bad, which corresponds to adding a guard $y \geq 3$ in the transition from q_7 to q_4. It is easy to verify that the TA with this added constraint is SNNI.

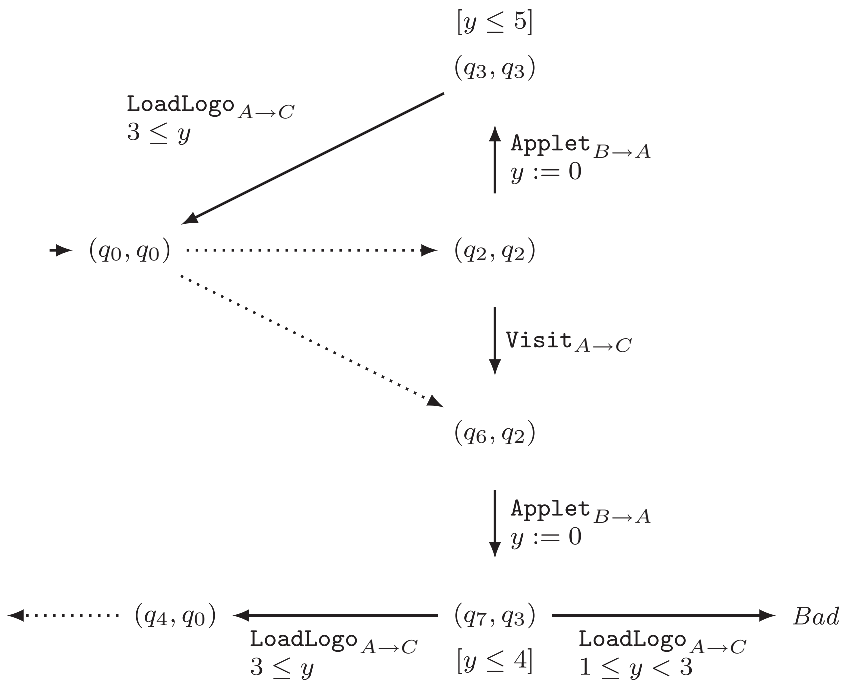


Figure 13. A part of $W \times_{\Sigma_l} K_2$ for the web privacy problem.

6. BSNNI and CSNNI control problems

In this section, we will show that for more restrictive non-interference properties (CSNNI and BSNNI), the control problem presents a major drawback: in the general case, there is no most permissive controller.

The CSNNI-CP (resp. BSNNI-CP) we are interested in is the following:

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

The CSNNI-CSP (resp. BSNNI-CSP) asks to compute a witness when the answer to the CSNNI-CP (resp. BSNNI-CSP) is 'yes'.

6.1 *CSNNI-CP and CSNNI-CSP*

Theorem 6.1: *For finite automata, the CSNNI-CP is in PTIME.*

Proof: Let A be a finite automaton, and we show that there exists a controller C such that $C(A)$ is CSNNI iff $A\backslash\Sigma_c$ is CSNNI.

The *if* direction is obvious: the controller $C_\forall$ that prevents any controllable action from occurring is defined by $C_\forall(\rho) = \varnothing$, $\forall \rho \in Runs(A)$. It is easy to see that $C_\forall(A)$ is isomorphic to $A\backslash\Sigma_c$ and thus bisimilar.

This *only if* direction is proved as follows: let A_1 and A_2 be two finite automata over alphabet Σ^ε such that A_1 weakly simulates A_2. Consider $A_1' = A_1\backslash\{e\}$ and $A_2' = A_2\backslash\{e\}$ for $e \in \Sigma$. Clearly, A_1' simulates A_2' (by definition of the simulation relation).

Therefore, if there exists C s.t. $C(A)$ is CSNNI, then so is $C(A)\backslash\Sigma'$ for any $\Sigma' \subseteq \Sigma$. It follows that $C(A)\backslash\Sigma_c$ must be CSNNI.

The CSNNI-CP reduces to the CSNNI-VP which is PTIME for finite automata. □

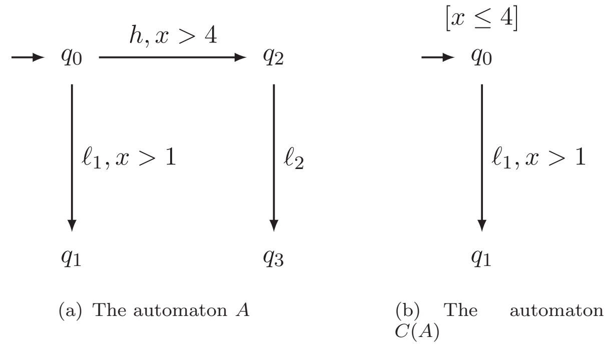
Theorem 6.2: *For the class of deterministic finite automata, the CSNNI-CSP is PSPACE-complete.*

Proof: By Lemma 3.11, for deterministic automata, SNNI is equivalent to CSNNI. Hence the CSNNI-CSP is equivalent to the SNNI-CSP which is PSPACE-complete by Theorem 5.21. □

In the timed setting, the previous reduction to a verification problem cannot be applied as illustrated by the following example.

Example 6.6: Let A be the deterministic timed automaton given in Figure 14 (a) with $\Sigma_l = \{\ell_1, \ell_2\}$, $\Sigma_h = \{h\}$ and $\Sigma_c = \{\ell_1\}$. $A\backslash\Sigma_c$ is neither CSNNI nor SNNI (here SNNI and CSNNI are equivalent since A is deterministic). However, there exists a controller C such that $C(A)$ is both CSNNI and SNNI. $C(A)$ can be given by the timed automaton given in Figure 14(b).

However, for the timed automata in dTA, thanks to Lemma 3.11 and Theorems 5.19 and 5.21, we have the following:


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[r, "{h, x>4}"] \arrow[d, "{\ell_1, x>1}"] & q_2 \arrow[d, "\ell_2"] \\ & q_1 & q_3 \end{tikzcd}$$
(a) The automaton A

$$\begin{tikzcd} \arrow[r] & q_0 \, [x \le 4] \arrow[d, "{\ell_1, x>1}"] \\ & q_1 \end{tikzcd}$$
(b) The automaton $C(A)$

Figure 14. Counterexample of theorem 6.1 in timed setting.

Theorem 6.4: *For timed automata in dTA, the CSNNI-CP and CSNNI-CSP are EXPTIME-complete.*

Proof: By Lemma 3.11, the CSNNI-CP/CSNNI-CSP is equivalent to the SNNI-CP/SNNI-CSP for dTA, and by Theorem 5.19, it follows that CSNNI-CP and CSNNI-CSP are EXPTIME-complete. □

Moreover, for dTA, thanks to the algorithm of Section 5, there always exists a most permissive controller for CSNNI. However, we will now show that there is a non-deterministic finite automaton s.t. there is no most permissive controller ensuring CSNNI.

Proposition 6.5: *There is no most permissive controller ensuring CSNNI for the finite automaton $A \notin dTA$ of Figure 5(a) (i.e. such that $A\backslash\Sigma_h$ is non-deterministic) with $\Sigma_h = \{h\}$, $\Sigma_l = \{\ell_1, \ell_2, \ell_3\}$ and $\Sigma_c = \{\ell_2, \ell_3\}$.*

Proof: Let A_c be the finite automaton of Figure 5(a) with $\Sigma_h = \{h\}$, $\Sigma_l = \{\ell_1, \ell_2, \ell_3\}$ and $\Sigma_c = \{\ell_2, \ell_3\}$. $A_c \notin dTA$ since $A_c\backslash\Sigma_h$ is non-deterministic. This automaton is not CSNNI. The controllers C_1 and C_2 of Figure 15 make the system CSNNI. However, $(C_1 \cup C_2)(A_c) = A_c$ is not CSNNI and, by construction is the only possible controller more permissive than C_1 and C_2. Therefore, there is no most permissive controller ensuring CSNNI for A_c with Σ_c. □

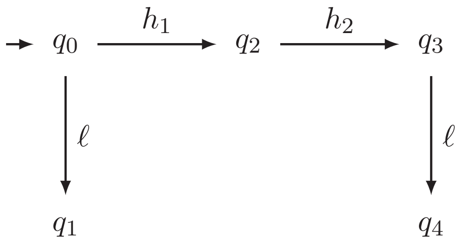
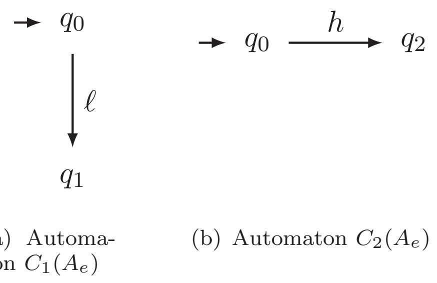

$$\begin{tikzcd} \arrow[r] & q_0 \arrow[r, "h_1"] \arrow[d, "\ell"] & q_2 \arrow[r, "h_2"] & q_3 \arrow[d, "\ell"] \\ & q_1 & & q_4 \end{tikzcd}$$

Figure 16. The automaton A_i.


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[d, "\ell"] \\ & q_1 \end{tikzcd}$$
(a) Automaton $C_1(A_e)$

$$\begin{tikzcd} \arrow[r] & q_0 \arrow[r, "h"] & q_2 \end{tikzcd}$$
(b) Automaton $C_2(A_e)$

Figure 17. Automata $C_1(A_e)$ and $C_2(A_e)$.

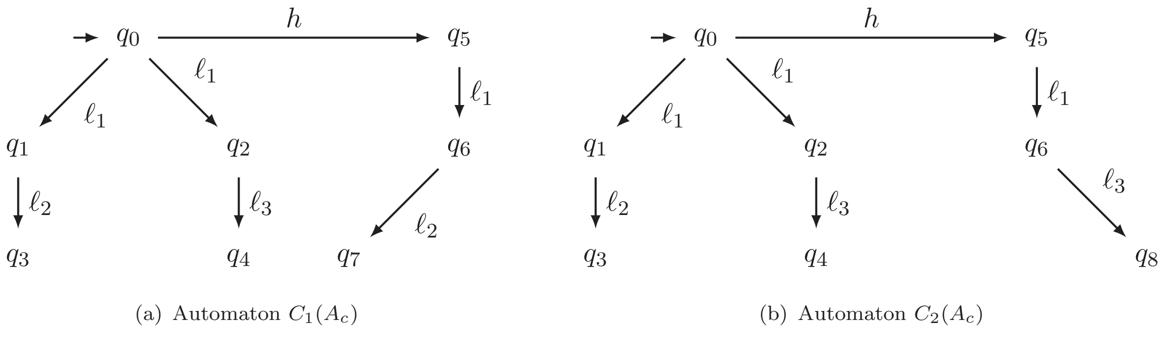
6.2 *BSNNI-CP and BSNNI-CSP*

We first show by Example 6.6 that even if there exists a controller for a finite automaton A and a controllable alphabet Σ_c ensuring BSNNI (i.e. the answer to BSNNI-CP is true), it is possible to have $A\backslash\Sigma_c$ not BSNNI.

Example 6.7: Let A_i be the finite automaton of Figure 16 with $\Sigma_h = \{h_1, h_2\}$ et $\Sigma_l = \{\ell\}$. This automaton is BSNNI, then the answer to BSNNI-CP is true for all Σ_c. However, for $\Sigma_c = \{h_2\}$, the automaton $A_i\backslash\Sigma_c = A_e$ is not BSNNI.

We will now prove that for deterministic finite automaton there is not always a most permissive controller that enforces BSNNI. This result is in contrast with CSNNI where a most permissive controller always exists for dTA.

Proposition 6.7: *There is no most permissive controller ensuring BSNNI for the deterministic finite automaton of Figure 6(a) with $\Sigma_h = \{h\}$, $\Sigma_l = \{\ell\}$ and $\Sigma_c = \{\ell, h\}$.*


$$\begin{tikzcd} & & q_0 \arrow[dl, "\ell_1"'] \arrow[dr, "\ell_1"] \arrow[rr, "h"] & & q_5 \arrow[d, "\ell_1"] \\ & q_1 \arrow[d, "\ell_2"] & & q_2 \arrow[d, "\ell_3"] & q_6 \arrow[dl, "\ell_2"] \\ & q_3 & & q_4 \quad q_7 & \end{tikzcd}$$
(a) Automaton $C_1(A_c)$

$$\begin{tikzcd} & q_0 \arrow[dl, "\ell_1"'] \arrow[dr, "\ell_1"] \arrow[rr, "h"] & & q_5 \arrow[d, "\ell_1"] \\ q_1 \arrow[d, "\ell_2"] & & q_2 \arrow[d, "\ell_3"] & q_6 \arrow[dr, "\ell_3"] \\ q_3 & & q_4 & & q_8 \end{tikzcd}$$
(b) Automaton $C_2(A_c)$

Figure 15. Automata $C_1(A_c)$ and $C_2(A_c)$.

Table 4. Summary of the results for CSNNI and BSNNI control problems.

	A Finite automaton	
	$A\backslash\Sigma_h$ Non-det.	$A\backslash\Sigma_h$ Det.
CSNNI-CP	PTIME (Theorem 6.1)	PTIME (Theorem 6.1)
CSNNI-CSP	NMPC* (Proposition 6.5)	PSPACE-C (Theorem 6.2)
BSNNI-CSP	NMPC* (Proposition 6.7)	NMPC* (Proposition 6.7)
	A Timed automaton	
	$A\backslash\Sigma_h$ Non-det.	$A\backslash\Sigma_h$ Det.
CSNNI-CP	Open	EXPTIME-C (Theorem 6.4)
CSNNI-CSP	NMPC* (Proposition 6.5)	EXPTIME-C (Theorem 6.4)
BSNNI-CSP	NMPC* (Proposition 6.7)	NMPC* (Proposition 6.7)

*NMPC means that there not always exists a most permissive controller.

Proof: Let A_e be the deterministic finite automaton of Figure 6(a) with $\Sigma_h = \{h\}$, $\Sigma_l = \{\ell\}$ and $\Sigma_c = \{\ell, h\}$. This automaton is not BSNNI. The controllers C_1 and C_2 of Figure 17 make the system BSNNI. However, $(C_1 \cup C_2)(A_e) = A_e$ is not BSNNI and, by construction is the only possible controller more permissive than C_1 and C_2. Therefore, there is no most permissive controller ensuring BSNNI for A_e with Σ_c. □

The summary of the results for CSNNI and BSNNI control problems is given in Table 4.

7. Conclusion and future work

In this paper, we have studied the SNNI-CP and 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 determinised 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 determinised; (3) there is not always a least restrictive (most permissive) controller for (bi)simulation-based non-interference even for untimed finite automata. However, there is a most permissive controller for CSNNI if $A\backslash\Sigma_h$ is deterministic and CSNNI-CP and CSNNI-CSP are EXPTIME-complete in this case in the timed setting.

The summary of the results is given in Tables 1 and 2 for the verification problems and Tables 3 and 4 for the control problems.

Our future work will first focus on the extension of these results to the more sophisticated notion of INI of Rushby (1992) for timed systems.

We will also investigate the CSNNI-CP (and BSNNI-CP) as even when there is no most permissive controller; it is interesting to find one. Finally, another future direction will consist of determining conditions under which a least restrictive controller exists for the BSNNI-CSP.

Notes

1. For $i = 0$, this is the amount of time since the system started.
2. We assume that $Q_1 \cap Q_2 = \varnothing$ and $X_1 \cap X_2 = \varnothing$.
3. Recall that A_2 is deterministic.

References

Alur, R., & Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science, 126*, 183–235.

Barbuti, R., & Tesei, L. (2003). A decidable notion of timed non-interference. *Fundamenta Informaticae, 54*, 137–150.

Barthe, G., Pichardie, D., & Rezk, T. (2007). A certified lightweight non-interference java bytecode verifier. In R. De Nicola (Ed.), *Proceedings of the 16th European conference on Programming (ESOP'07)* (pp. 125–140). Braga: Springer-Verlag.

Benattar, G., Cassez, F., Lime, D., & Roux, O.H. (2009, September). Synthesis of non-interferent timed systems. In J. Ouaknine & F.W. Vaandrager (Eds.), *Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, *Lecture Notes in Computer Science* (Vol. 5813, pp. 28–42). Budapest: Springer.

Bortz, A., & Boneh, D. (2007). Exposing private information by timing web applications. In P. Patel-Schneider & P. Shenoy (Eds.), *Proceedings of the 16th International Conference on World Wide Web (WWW '07)* (pp. 621–628). Banff, Alberta, Canada: ACM.

Bossi, A., Piazza, C., & Rossi, S. (2007). Compositional information flow security for concurrent programs. *Journal of Computer Security, 15*, 373–416.

Bouyer, P., D'Souza, D., Madhusudan, P., & Petit, A. (2003, July). Timed control with partial observability. In W.H. Jr & F. Somenzi (Eds.), *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, *Lecture Notes in Computer Science* (Vol. 2725, pp. 180–192). Boulder, CO: Springer.

Cassez, F. (2009, June). The dark side of timed opacity. In J. H. Park, H.-H. Chen, M. Atiquzzaman, C. Lee, T, Kim, & S.-S. Yeo (Eds.), *Proceedings of the 3rd International Conference on Information Security and Assurance (ISA'09)*, *Lecture Notes in Computer Science* (Vol. 5576, pp. 21–30). Seoul: Springer.

Cassez, F., Dubreil, J., & Marchand, H. (2009, October). Dynamic observers for the synthesis of opaque systems. In Z. Liu & A.P. Ravn (Eds.), *7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*, *Lecture Notes in Computer Science* (Vol. 5799, pp. 352–367). Macao, China: Springer.

Cassez, F., Dubreil, J., & Marchand, H. (2012). Synthesis of opaque systems with static and dynamic masks. *Formal Methods in System Design, 40*, 88–115.

Cassez, F., Mullins, J., & Roux, O.H. (2007). Synthesis of non-interferent systems. In V. Gorodetsky, I. Kotenko, & V. Skormin (Eds.), *4th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07)*, *Communications in Computer and Information Science* (Vol. 1, pp. 307–321). St. Petersburg, Russia: Springer.

Čerāns, K. (1992). Decidability of bisimulation equivalence for parallel timer processes. In G. von Bochmann & D.K. Probst (Eds.), *Proceedings of the Fourth Workshop on Computer-Aided Verification (CAV'92)*, *Lecture Notes in Computer Science* (pp. 302–315). Montreal, Canada: Springer-Verlag.

D'Souza, D., & Madhusudan, P. (2002). Timed control synthesis for external specifications. In H. Alt & A. Ferreira (Eds.), *19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02)*, *Lecture Notes in Computer Science* (Vol. 2285, pp. 571–582). Antibes-Juan les Pins, France: Springer.

D'Souza, D., Raghavendra, K.R., & Sprick, B. (2005). An automata based approach for verifying information flow properties. *Electronic Notes in Theoretical Computer Science, 135*, 39–58.

Felten, E.W., & Schneider, M.A. (2000). Timing attacks on Web privacy. In P. Samarati (Ed.), *CCS '00: Proceedings of the 7th ACM Conference on Computer and Communications Security, Athens* (pp. 25–32). New York, NY: ACM Press.

Finkel, O. (2005). On decision problems for timed automata. *Bulletin of the European Association for Theoretical Computer Science, 87*, 185–190.

Focardi, R., Ghelli, A., & Gorrieri, R. (1997, September). Using non-interference for the analysis of security protocols. In H. Orman & C. Meadows (Eds.), *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*. Piscataway, NJ: Rutgers University.

Focardi, R., & Gorrieri, R. (1997). The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering, 23*, 550–571.

Focardi, R., & Gorrieri, R. (2001). Classification of security properties (Part I: Information flow). In R. Focardi & R. Gorrieri (Eds.), *Foundations of Security Analysis and Design I: FOSAD 2000 Tutorial Lectures*, *Lecture Notes in Computer Science* (Vol. 2171, pp. 331–396). Heidelberg: Springer-Verlag.

Gardey, G., Mullins, J., & Roux, O.H. (2005, August). Non-interference control synthesis for security timed automata. In M. Backes & A. Scedrov (Eds.), *3rd International Workshop on Security Issues in Concurrency (SecCo'05)*, *Electronic Notes in Theoretical Computer Science* (pp. 33–53). San Francisco, CA: Elsevier.

Hadj-Alouane, N.B., Lafrance, S., Lin, F., Mullins, J., & Yeddes, M. (2005a). Characterizing intransitive noninterference for 3-domain security policies with observability. *IEEE Transaction on Automatic Control, 50*, 948–958.

Hadj-Alouane, N.B., Lafrance, S., Lin, F., Mullins, J., & Yeddes, M. (2005b). On the verification of intransitive noninterference in multilevel security. *IEEE Transactions on Systems, Man and Cybernetics, 35*, 948–958.

Henzinger, T., & Kopke, P. (1997). Discrete-time control for rectangular hybrid automata. In P. Degano, R. Gorrieri, & A. Marchetti-Spaccamela (Eds.), *International Colloquium on Automata, Languages and Programming (ICALP'97)* (pp. 582–593). Bologna, Italy: Santa Barabara.

Kammuller, F. (2008). Formalizing non-interference for a simple bytecode language in Coq. *Formal Aspects of Computing, 20*, 259–275.

Kocher, P.C. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz (Ed.), *International Cryptology Conference (CRYPTO'96)* (pp. 104–113). Santa Barabara, CA: Springer-Verlag.

Kotcher, R., Pei, Y., Jumde, P., & Jackson, C. (2013). Cross-origin pixel stealing: timing attacks using CSS filters. In V. Gligor & M. Yung (Eds.), *Proceedings of the 2013 ACM SIGSAC conference on computer and communications security* (pp. 1055–1062). Berlin: ACM.

Krohn, M., & Tromer, E. (2009). Noninterference for a practical DIFC-based operating system. In A. Myers & D. Evans (Eds.), *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy* (pp. 61–76). Washington, DC: IEEE Computer Society.

Kupferman, O., & Vardi, M. (1997). Synthesis with incomplete information. In H. Barringer, M. Fisher, D. Gabbay & G. Gough (Eds.), *Proceedings of the 2nd International Conference on Temporal Logic (ICTL'97)* (pp. 91–106). Manchester, UK: Kluwer.

Lamouchi, H., & Thistle, J. (2000). Effective control synthesis for DES under partial observations. In *IEEE Conference on Decision and Control* (pp. 22–28). Sydney, Australia: IEEE.

Laroussinie, F., & Schnoebelen, P. (2000). The state-explosion problem from trace to bisimulation equivalence. In J. Tiuryn (Ed.), *Foundations of Software Science and Computation Structures (FoSSaCS 2000)*, *Lecture Notes in Computer Science* (Vol. 1784, pp. 192–207). Berlin, Germany: Springer-Verlag.

Lin, F., & Wonham, W. (1988). On observability of discrete-event systems. *Information Sciences, 44*, 173–198.

Lin, F., & Wonham, W. (1995). Supervisory control of timed discrete-event systems under partial observation. *IEEE Transactions on Automatic Control, 40*, 558–562.

Maler, O., Pnueli, A., & Sifakis, J. (1995). On the synthesis of discrete controllers for timed systems. In E.W. Mayr, & C. Puech (Eds.), *Annual Symposium on Theoretical Aspects of Computer Science (STACS '95)* (pp. 229–242). Munich, Germany: Springer.

Mazaré, L. (2004). Using unification for opacity properties. In *Workshop on Issues in the Theory of Security (WITS'04)* (pp. 165–176). Barcelona, Spain.

Moez, Y., Lin, F., & Ben Hadj-Alouane, N. (2009). Modifying security policies for the satisfaction of intransitive non-interference. *IEEE Transactions on Automatic Control, 54*, 1961–1966.

R. Focardi, R.G., & Martinelli, F. (2003). Real-time information flow analysis. *IEEE Journal on Selected Areas in Communications, 21*, 20–35.

Rushby, J. (1992). *Noninterference, transitivity and channel-control security policies* (Technical Report CSL-92-02). Menlo Park, CA: SRI International.

Sabelfeld, A., & Myers, A. (2003). Language-based information-flow security. *IEEE Journal on Selected Areas in Communications, 21*, 1–15.

Saboori, A., & Hadjicostis, C. (2008). Opacity-enforcing supervisory strategies for secure discrete event systems. In *the 47th IEEE Conference on Decision and Control* (pp. 889–894). Cancun, Mexico: IEEE.

Stockmeyer, L.J., & Meyer, A.R. (1973). Word problems requiring exponential time: Preliminary report. In A.V. Aho, A.B. Borodin, R.L Constable, R.W. Floyd, M.A. Harrison, R.M.

Karp, & H.R. Strong (Eds.), *ACM Symposium on Theory of Computing (STOC'73)* (pp. 1–9). San Diego, CA: ACM.

Tasiran, S., Alur, R., Kurshan, R.P., & Brayton, R.K. (1996). Verifying abstractions of timed systems. In U. Montanari & V. Sassone (Eds.), *Conference on Concurrency Theory (CONCUR'96)*, *Lecture Notes in Computer Science* (Vol. 1119, pp. 546–562). Pisa, Italy: Springer.

van der Meyden, R., & Zhang, C. (2006). Algorithmic verification of noninterference properties. In M. Beek & F. Gadducci (Eds.), *Proceedings of the Second International Workshop on Views on Designing Complex Architectures (VODCA 2006)*, *Electronic Notes in Theoretical Computer Science* (Vol. 168, pp. 61–75). Amsterdam, The Netherlands: Elsevier.