Fundamenta Informaticae 34 (2002) 1–12
IOS Press

A Notion of Non-Interference for Timed Automata

Roberto Barbuti
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40 - 56125 Pisa - Italy
email: barbuti@di.unipi.it

Nicoletta De Francesco
Dipartimento di Ingegneria dell'Informazione, Università di Pisa
Via Diotisalvi, 2 - 56126 Pisa - Italy
email: nicoletta.defrancesco@iet.unipi.it

Antonella Santone
Facoltà di Ingegneria, Università del Sannio
Palazzo Bosco Lucarelli, Piazza Roma - 82100 Benevento - Italy
email: santone@unisannio.it

Luca Tesei
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40 - 56125 Pisa - Italy
email: tesei@di.unipi.it

Abstract. The non-interference property of concurrent systems is a security property concerning the flow of information among different levels of security of the system. In this paper we introduce a notion of timed non-interference for real-time systems specified by timed automata. The notion is presented using an automata based approach and then it is characterized also by operations and equivalence between timed languages. The definition is applied to an example of a time-critical system modeling a simplified control of an airplane.

1. Introduction

The non-interference property, introduced in [10], is a security property concerning the flow of information among different levels of security present in the system. Suppose, for simplicity, that the security levels are two: *high* and *low*. Thus, the behaviors of the system are divided into two classes: the high-level behaviors executed by high-level users and the low-level behaviors executed by low-level users. Moreover, the system can be observed by users at different levels. In particular high-level behaviors should not be visible to low-level users. The system respects the non-interference property if the low-level behaviors are not affected by the high-level ones. In other words, if a system P acts in an environment where low-level and high-level users are present and are doing all that they can do, P is secure if the observations that P offers to the low-level users when high-level behaviors are present and hidden are equal, in some suitable sense, to the ones that P offers when no high-level behavior is present.

The initial idea has been fruitful in the framework of security, especially in the 90's. It has been applied, for instance, to non-deterministic systems described by a CCS-like process algebra [7, 8, 5], to the analysis of security protocols [6], to probabilistic systems [15] and to information flows in a timed process algebra with discrete time domain [9]. Different notions of non-interference proposed in the literature have been compared and summarized in [5, 12]. An overview of some important questions concerning non-interference definitions, in a CSP setting, can be found in [13].

In this paper we define a notion of non-interference for real-time systems, where the time is a crucial parameter. We represent these systems by timed automata [2], which are a well-known formalism to describe qualitative and quantitative time constraints on systems. Timed automata have been widely studied also for their possible use in the verification of real-time systems [3, 1, 16, 11].

To define our notion of non-interference, we partition the alphabet of the timed automaton, representing a real-time system, in two classes: high-level actions and low-level actions. Our notion of timed non-interference depends on a natural number n representing a minimum delay between high-level actions such that the low-level behaviors are not affected by the high-level ones. Thus, this notion is suitable to detect interference due to high frequency of high-level actions. An intuitive example of this interferences is the following. Consider a timed automaton T that models a speed-dependent real-time system like an airplane control system. It has to control a lot of basic events and has to respond with basic actions in order to maintain, say, the flight stability. These actions and events may be considered low-level actions and are always activated. When the pilot decides, for example, to turn right, he uses the cloche and this may correspond to an occurrence of a high-level action. Thus, the system receives high-level events (in this case cloche movements signals) separated by certain delays; *it must respond to them and must continue to catch and manage basic events.* When this happens we say that high-level actions delays magnitude does not affect the basic behavior of the system. Intuitively if the cloche movements signals are sent to the automaton with a too much high frequency, it could reach a state in which it is no longer able to manage basic events.

We use an automata based approach to define the non-interference property of a system and then we give also a timed language characterization to justify more formally the operations defined by means of certain special automata and of a simplified product between timed automata. The definition of the timed non-interference is given using timed automata equivalence which is a suitable equivalence for our purposes, but it is not effectively computable. We discuss this problem and define some possible ways to address it in Section 5.

The paper is organized as follows: Section 2 recalls timed automata, Section 3 defines the timed non-interference for timed automata, Section 4 shows an example, and Section 5 discusses the results and future work.

2. Timed Automata

In this section we recall the definition of timed automata [2]. In the following, $\mathcal{R}$ is the set of real numbers and $\mathcal{R}^+$ the set of non-negative real numbers. A *clock* takes values from $\mathcal{R}^+$. Given a set $\mathcal{X}$ of clocks, a *clock valuation* over $\mathcal{X}$ is a function assigning a non-negative real number to every clock. The set of valuations of $\mathcal{X}$, denoted $\mathcal{V}_\mathcal{X}$, is the set of total function from $\mathcal{X}$ to $\mathcal{R}^+$. Given $\nu \in \mathcal{V}_\mathcal{X}$ and $\delta \in \mathcal{R}^+$, with $\nu + \delta$ we denote the valuation that maps each clock $x \in \mathcal{X}$ into $\nu(x) + \delta$.

Given a set $\mathcal{X}$ of clocks, a *reset* γ is a subset of $\mathcal{X}$. The set of all resets of $\mathcal{X}$ is denoted by $\Gamma_\mathcal{X}$. Given a valuation $\nu \in \mathcal{V}_\mathcal{X}$ and a reset γ, with $\nu\backslash\gamma$ we denote the valuation

$$\nu\backslash\gamma(x) = \begin{cases} 0 & \text{if } x \in \gamma \\ \nu(x) & \text{if } x \notin \gamma \end{cases}$$

Given a set $\mathcal{X}$ of clocks, the set $\Psi_\mathcal{X}$ of *clock constraints* over $\mathcal{X}$ are defined by the following grammar:

$$\psi ::= true \mid false \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid x\#t \mid x - y\#t$$

where $x, y \in \mathcal{X}$, $t \in \mathcal{R}^+$, and $\#$ is a binary operator in $\{<, >, \leq, \geq, =\}$. Clock constraints are evaluated over clock valuations. The satisfaction by a valuation $\nu \in \mathcal{V}_\mathcal{X}$ of the clock constraint $\psi \in \Psi_\mathcal{X}$ is denoted by $\nu \models \psi$ and it is defined as follows:

$\nu \models true$ and $\nu \not\models false$

$\nu \models \psi_1 \wedge \psi_2$ iff $\nu \models \psi_1 \wedge \nu \models \psi_2$

$\nu \models \psi_1 \vee \psi_2$ iff $\nu \models \psi_1 \vee \nu \models \psi_2$

$\nu \models \neg\psi$ iff $\nu \not\models \psi$

$\nu \models x\#t$ iff $\nu(x)\#t$

Definition 2.1. (Timed automaton) A timed automaton T is a tuple $(Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$, where: Q is a finite set of states, Σ is a finite alphabet of actions, $\mathcal{E}$ is a finite set of edges, $I \subseteq Q$ is the set of initial states, $R \subseteq Q$ is the set of repeated states, $\mathcal{X}$ is a finite set of clocks. Each edge $e \in \mathcal{E}$ is a tuple in $Q \times \Psi_\mathcal{X} \times \Gamma_\mathcal{X} \times \Sigma \times Q$.

If $e = (q, \psi, \gamma, \sigma, q')$ is an edge, q is the *source*, q' is the *target*, ψ is the *constraint*, σ is the *label*, γ is the *reset*.

The semantics of a timed automaton T is an infinite transition system $\mathcal{S}(T) = (S, \rightarrow)$, where S is a set of states and $\rightarrow$ is the transition relation. The states S of $\mathcal{S}(T)$ are pairs (q, ν), where $q \in Q$ is a state of T, and ν is a valuation. An initial state of $\mathcal{S}(T)$ is a state (q, ν), where $q \in I$ is an initial state of T and ν is the valuation which assigns 0 to every clock in $\mathcal{X}$. At any state q, given a valuation ν, T can stay idle or it can perform an action labeling an outgoing edge e. The rules to derive the transitions of $\mathcal{S}(T)$ are the following:

$$1.\ \frac{\delta \in \mathcal{R}^+}{(q,\nu) \xrightarrow{\delta} (q, \nu + \delta)} \qquad 2.\ \frac{(q,\psi,\gamma,\sigma,q') \in \mathcal{E}, \nu \models \psi}{(q,\nu) \xrightarrow{\sigma} (q', \nu \backslash \gamma)}$$

Rule 1. represents the case in which T stays idle in a state and the time passes, while Rule 2. corresponds to the occurrence of an action.

Definition 2.2. (run, action sequence) Given a timed automaton $T = (Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$, a *run* of the automaton is an infinite sequence of states and transitions of $\mathcal{S}(T)$ $s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots$ where

- $s_0 = (q, \nu)$, $q \in I$ and $\nu(x) = 0$ for every $x \in \mathcal{X}$
- a state $q \in R$ exists such that q occurs infinitely often in the pairs of the sequence $\{s_i\}_{i \in I\!N}$

Note that, given a run $s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots$, for each i, $l_i \in (\Sigma \cup \mathcal{R}^+)$.
Let r be a run.

- The *time sequence* $\overline{t_j}$ of the time elapsed from state s_0 to state s_j in r is defined as follows:

$$t_0 = 0$$
$$t_{i+1} = t_i + \begin{cases} 0 & \text{if } l_i \in \Sigma \\ l_i & \text{otherwise} \end{cases}$$

- The *event sequence* of the events occurring during r, including the elapsed times, is defined as follows: $(l_0, t_0)(l_1, t_1) \ldots$
- The *action sequence* of r is the projection of the event sequence of r on the pairs $\{(l, t) | l \in \Sigma\}$

Definition 2.3. (timed word, timed language) Let Σ be an alphabet. A *timed word* over Σ is an infinite sequence $\overline{(\sigma, t)} = (\sigma_0, t_0)(\sigma_1, t_1) \ldots$, where $\sigma_i \in \Sigma$, $t_i \in \mathcal{R}^+$ and $t_i \leq t_{i+1}$, for all $i = 0, 1, \ldots$.

A *timed language* over Σ is a subset of the set of all timed words over Σ.

Definition 2.4. (acceptance) Given a timed automaton $T = (Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$, a timed word w over Σ is *accepted* by T iff there exists a run r of T, whose action sequence is v, such that $w = v$. The set of timed words accepted by T is called the *accepted language* of T and is denoted by $\mathcal{L}(T)$.

Note that we use the Büchi acceptance condition for the runs [2]. Moreover, we say that two timed automata T_1 and T_2 are *equivalent*, we write $T_1 \approx T_2$, if $\mathcal{L}(T_1) = \mathcal{L}(T_2)$.

Automata with ϵ-edges are defined in the same way, but with a special action, the non-observable action ϵ, belonging to Σ. Thus some transitions of the semantic automaton $\mathcal{S}(T)$ can be labeled by ϵ, and they are called ϵ-transitions. When defining the accepted language for an automaton with ϵ-edges, we must consider the action sequences as the the projection of the event sequences on the pairs $\{(l,t) \mid l \in \Sigma - \{\epsilon\}\}$.

The design of complex systems can be simplified by modeling subsystems with different timed automata and considering, for the whole system, a suitable product of them. Now we define a product operation which is a syntactic operation between timed automata.

Definition 2.5. (Product) Let $\mathrm{T}_1 = (Q_1, \Sigma_1, \mathcal{E}_1, I_1, R_1, \mathcal{X}_1)$ and $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, I_2, R_2, \mathcal{X}_2)$ be two timed automata with $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. The product of T_1 and T_2, denoted by $T_1 \parallel T_2$, is the following timed automaton:

$$T_1 \parallel T_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \mathcal{E}, I_1 \times I_2, R_1 \times R_2, \mathcal{X}_1 \cup \mathcal{X}_2 \rangle$$

where $\mathcal{E}$ is defined by:

1. **Synchronization actions**

   $\forall \sigma \in \Sigma_1 \cap \Sigma_2, \forall (q_1, \psi_1, \gamma_1, \sigma, q_1') \in \mathcal{E}_1, \forall (q_2, \psi_2, \gamma_2, \sigma, q_2') \in \mathcal{E}_2$

   $\mathcal{E}$ contains $((q_1, q_2), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2'))$
2. T_1 **actions**

   $\forall \sigma \in \Sigma_1 \backslash \Sigma_2, \forall (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_1, \forall s \in Q_2$

   $\mathcal{E}$ contains $((q, s), \psi, \gamma, \sigma, (q', s)$
3. T_2 **actions**

   $\forall \sigma \in \Sigma_2 \backslash \Sigma_1, \forall (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_2, \forall s \in Q_1$

   $\mathcal{E}$ contains $((s, q), \psi, \gamma, \sigma, (s, q'))$

Thus, the product automaton behavior is the interleaving of the components behaviors where actions with the same name are executed synchronously.

Note that the product operation above is not the standard parallel composition on timed automata with Büchi acceptance condition. In that case the resulting automaton is defined in order to accept only the behaviors in which all the components have a run with respect to their sets of repeated states (see [2]). For our purposes it is sufficient to define, as above, a product that is simply the product between the timed transition tables (i.e. the automata without the set of repeated states needed for the acceptance condition) of the involved automata, and to set the set of repeated states of the resulting timed transition table to the cartesian product of the repeated states of the components. This would not work correctly in the general case, but here it is correct because we use the product only to restrict the acceptance of some timed words by a given timed automaton.

3. Non-Interference for Timed Automata

Non-interference for concurrent non-real-time systems has been modeled in process algebras using bisimulation equivalence (see, for example, [14, 7, 8, 13]): if the actions of a system P are divided into high-level and low-level ones, *the system respects the non-interference property if its behavior in absence of high-level actions is equivalent to its behavior, observed on low-level actions, when high-level actions occur*. To compare the two behaviors we check equivalence of two processes obtained from P: in the first one high-level actions are forbidden, in the second one they are hidden. Different notions of bisimulation equivalence are used to model different notions of non-interference. These have been reformulated in [9] in a real-time setting using a timed process algebra with a discrete time domain.

In this paper we define a new notion of non-interference using timed automata, a well-known formalism, with a dense time domain, for real-time systems modeling and verification. The notion is based on high-level actions delays magnitude and on equivalence of timed automata. Given a natural number n, we say that *high-level actions do not interfere with the system, considering a minimum delay n, if the system behavior in absence of high-level actions is equivalent to the system behavior, observed on low-level actions, when high-level actions can occur, but the delay between any two of them is greater than or equal to n*. Thus, if the environment of the system does not offer high-level events separated by less than n time units and the property holds, there is no way for low-level users to detect any high-level activity. The main improvement with respect to the untimed notion of non-interference is that time is observable and the property captures those systems in which the time delay between high-level actions *cannot* be used to construct illegal information flows from high-level to low-level users.

Let T be a timed automaton over the alphabet Σ. We suppose that Σ is partitioned into two disjoint sets of actions H and L: H is the set of high-level actions, while L is the set of low-level ones.

To observe the behavior of an automaton T in absence of high-level actions, we can compose T in parallel with an automaton, from now on called Inhib_H, that does not allow the execution of high-level actions.

The automaton Inhib_H is shown in Figure 1. In figures we use some usual conventions: a state with a dangling ingoing arc is an initial state, a state depicted with double circles belongs to the set of repeated states and an arc having as label a set of action (as L or H for instance) represents a set of edges, one for each action in the set (L or H), with the same clock constraint and clock reset. In the product $T \parallel \mathrm{Inhib}_H$, where all actions are synchronization actions, the component T cannot have a transition labeled by $\sigma \in H$ because his partner in synchronization, Inhib_H, never performs high-level actions (its constraints on high level actions are *false*). Thus only low-level actions are executed.

We have that $\mathcal{L}(T \parallel \mathrm{Inhib}_H)$ contains all basic behaviors of T, i.e. all timed words obtained by runs in which high-level actions do not occur.

Consider the automaton Interf_H^n in Figure 2. This automaton allows the execution of high-level actions only when they are separated by at least n time units. To see this, consider its

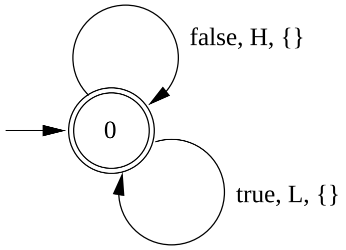

Figure 1. Inhib_H

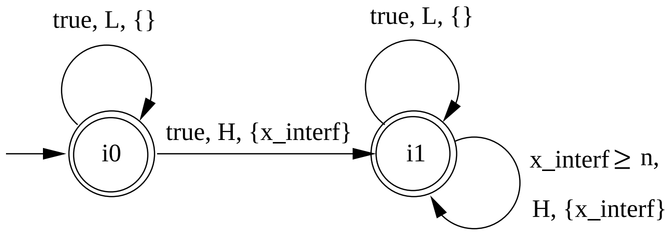

Figure 2. The structure of Interf_H^n

behavior from the initial state i_0. There, it can perform low-level actions without restrictions. If never high-level actions occur in the run, then it stays forever in i_0 cycling on low-level actions. If a high-level action occurs the automaton changes its state to i_1 and reset a clock called x_{interf}. Note that this clock is reset by all high-level actions. This means that, in state i_1, x_{interf} always records the time elapsed from the previous high-level action occurred. In state i_1 low-level actions are allowed again with no restrictions, but any high-level action can be executed only if at least n time units have elapsed from the previous one. This is expressed by the constraint $x_{\mathrm{interf}} \geq n$.

Given an automaton T, the product of T with Interf_H^n, $T \parallel \mathrm{Interf}_H^n$, allows to observe the set of all behaviors of T such that high-level actions occur at times separated by an interval whose length is greater than, or equal to n. Of course, we are assuming that T does not reset the clock x_{interf}.

Before giving the definition of n-non-interference, we need the following operation, which, if applied to an automaton T, returns an automaton having the same behaviors of T, but where high-level actions are non-observable.

Definition 3.1. (hiding of high-level actions) Let T be a timed automaton over an alphabet $\Sigma = (H, L)$. We denote by T/H the automaton obtained by T by replacing each edge $(q, \psi, \gamma, \sigma, q')$ of T with $\sigma \in H$, with the edge $(q, \psi, \gamma, \epsilon, q')$.

Now we can define formally the intuitive notion of non-interference given above.

Definition 3.2. (n-non-interference) Let T be a timed automaton over an alphabet $\Sigma = (H, L)$, and let $n \in \mathbb{N}$. High-level actions do not interfere in T with a minimum delay n

(equivalently we say that T is n-non-interfering), if and only if

$$(T \parallel \mathrm{Interf}_H^n)/H \;\approx\; T \parallel \mathrm{Inhib}_H \tag{1}$$

3.1. Characterization with Timed Languages

In this section we characterize the notion of timed non-interference, defined in the previous section, using timed languages. We define three operations on timed languages and then we relate them to the corresponding operations on automata. This provides a more formal justification of the definition of n-non-interference of the previous section. As a matter of fact, it was given following some remarks on the behaviors of the automata Inhib_H and Interf_H^n when composed with the automaton T representing the system. The proofs of the following propositions are rather simple. We only give the proof of Proposition 3.2. The other ones are similar.

First, consider the restriction of a timed language to low-level actions.

Definition 3.3. (restriction to low-level actions) Let I be a set of timed words over an alphabet $\Sigma = (H, L)$ (i.e. a timed language). $I|_L$ is the subset of I such that all elements are timed words on actions in L only:

$$I|_L = \{\overline{(\sigma,t)} \in I \mid \forall (\sigma_i, t_i) \in \overline{(\sigma,t)}.\ \sigma_i \in L\}$$

Proposition 3.1. *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$. Then $\mathcal{L}(T)|_L = \mathcal{L}(T \parallel \mathrm{Inhib}_H)$.* □

The restriction to high level actions separated by at least n time units is defined as follows.

Definition 3.4. (n-delay restriction) Let I be a set of timed words over an alphabet $\Sigma = (H, L)$. Let n be a natural number. I_H^n is the subset of I containing all timed words in I such that the delay between any two actions in H is at least n:

$$I_H^n = \{\overline{(\sigma,t)} \in I \mid \forall (\sigma_i, t_i), (\sigma_j, t_j) \in \overline{(\sigma,t)}.\ i \neq j \wedge \sigma_i, \sigma_j \in H \Rightarrow |t_i - t_j| \geq n\} \tag{2}$$

Proposition 3.2. *Let $T = (Q, \Sigma = (H, L), \mathcal{E}, I, R, X)$ be a timed automaton. Then $\mathcal{L}(T)_H^n = \mathcal{L}(T \parallel \mathrm{Interf}_H^n)$.*

Proof:
Let w be a timed word in $\mathcal{L}(T)_H^n$. This means that $w \in \mathcal{L}(T)$ and it satisfies the condition expressed in (2). Since w is accepted by T, there exists a run $r = (q_0, \nu_0) \xrightarrow{l_0} (q_1, \nu_1) \xrightarrow{l_1} \ldots$ whose action sequence equals w (actually, there exist infinitely many runs having this property; we take just one of them). Let $\overline{q}$ be a state of T, belonging to R, which is entered infinitely many times along r according to the Büchi acceptance condition.

We want to construct, from r, a run of $T \parallel \mathrm{Interf}_H^n$ whose action sequence equals w. There are two cases. The first one is when a high level action is never executed along r. In this case the run is just $r' = ((q_0, i_0), \nu_0 \cup \{x_{\mathrm{interf}} = 0\}) \xrightarrow{l_0} ((q_1, i_0), \nu_1^*) \xrightarrow{l_1} \ldots ((q_i, i_0), \nu_i^*) \xrightarrow{l_i} \ldots$ where

ν_i^* are the clock valuations ν_i of r extended with the valuation for the clock x_{interf} of Interf_H^n. Along r (and also r') this clock is never reset and Interf_H^n stays forever in state i_0 which is a repeated state. Thus, along the run so constructed, the state $(\overline{q}, i_0)$ is taken infinitely many times satisfying the acceptance condition; moreover r' has the same event sequence of r and thus the corresponding accepted timed word is w.

The second case is when at least one high-level action h is executed along r. Suppose this happens at step k. Consider the prefix of a run $r' = ((q_0, i_0), \nu_0 \cup \{x_{\text{interf}} = 0\}) \xrightarrow{l_0} ((q_1, i_0), \nu_1^*) \xrightarrow{l_1} \cdots ((q_k, i_0), \nu_k^*) \xrightarrow{l_k = h} ((q_{k+1}, i_1), \nu_{k+1} \cup \{x_{\text{interf}} = 0\})$ where ν_i^* are again the extensions of ν_i of r to handle the clock x_{interf}. If all actions following l_k in r are low-level, we can extend r' as in the first case and we have that the state $(\overline{q}, i_1)$ occurs infinitely many times in r', and so r' is a run of $T \parallel \text{Interf}_H^n$. If, on the other hand, other high-level actions occur in r, i.e. $r = \cdots (q_{k+1}, \nu_{k+1}) \xrightarrow{l_{k+1}} \cdots (q_j, \nu_j) \xrightarrow{l_j} (q_{j+1}, \nu_{j+1}) \cdots$, with $l_j \in H$ and for each $l_i, k < i < j$, $l_i \in L \cup \mathcal{R}^+$, we can extend r' as follows: $r' = \cdots ((q_{k+1}, i_1), \nu_{k+1}^*) \xrightarrow{l_{k+1}} \cdots ((q_j, i_1), \nu_j^*) \xrightarrow{l_j} (q_{j+1}, \nu_{j+1}^*) \cdots$ Since r satisfies the condition of (2), then at least n time units have elapsed between the occurrence of l_k and that of l_j. Now, since $\nu_{k+1}^* = \nu_{k+1} \cup \{x_{\text{interf}} = 0\}$ and x_{interf} is not reset by low-level actions, then the value held by x_{interf} in ν_j^* is greater than, or equal to, n, and this means that r' is a prefix of a run of $T \parallel \text{Interf}_H^n$. Since $\nu_{j+1}^* = \nu_{j+1} \cup \{x_{\text{interf}} = 0\}$, the same argumentation can be used to show that the next high-level action of r (if any) can be executed in r', and so on. Thus, $w \in \mathcal{L}(T \parallel \text{Interf}_H^n)$. The converse can be easily proven by a similar argument. □

The hiding of high-level actions is expressed as follows.

Definition 3.5. (hiding of high-level actions) Let I be a set of timed words over an alphabet $\Sigma = (H, L)$. I/H contains the timed words $\overline{(\sigma, t)}$ of I in which the pairs (σ_i, t_i) with $\sigma_i \in H$ are discarded:

$$I/H = \left\{ \omega' \;\middle|\; \begin{array}{l} \omega = \overline{(\sigma,t)} \in I \text{ and } \omega' \text{ is the projection of } \omega \\ \text{on the pairs } \{(\sigma,t) | \sigma \in L\} \end{array} \right\}$$

Proposition 3.3. *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$. Then $\mathcal{L}(T)/H = \mathcal{L}(T/H)$.* □

The following proposition states that the above characterization correctly expresses n-non-interference.

Theorem 3.1. *Let T be a timed automaton over an alphabet $\Sigma = (H, L)$, and $n \in I\!N$.*

$$T \textit{ is } n\textit{-non-interfering} \quad \textit{iff} \quad \mathcal{L}(T)_H^n/H = \mathcal{L}(T)|_L$$

Proof:
Follows from Definition 3.2 and the three propositions above. □

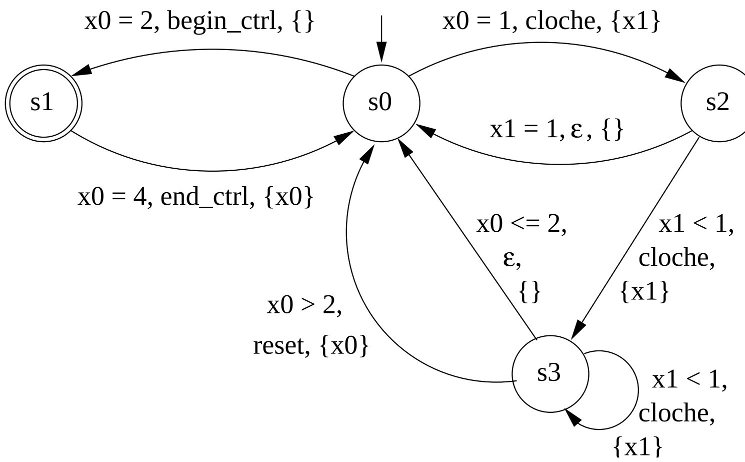

Figure 3. Automaton T: a simplified airplane control

4. An example

In this section we consider a simple control T of an airplane (Figure 3) and study its non-interference properties. The system periodically executes, at predefined instants, a set of operations to control flight stability. The control operations begin with action *begin_ctrl* and end with action *end_ctrl*. State *s1* is an abstraction for the control operations, which require 2 time units to be completed. One time unit before entering each control cycle the system can catch an input action from the pilot; in this simple case we consider only a single input action *cloche* modeling cloche movements. When *cloche* occurs, the system handles it and then continues to manage control actions. Let *begin_ctrl* and *end_ctrl* be low-level actions and *cloche* and *reset* be the high-level actions.

At first consider the behavior of T when the high-level action *cloche* is disabled (i.e. the moves of $T \parallel \mathrm{Inhib}_H$); this consists in a simple cycle between states *s0* and *s1* where transitions are separated by exactly 2 time units.

Thus $\mathcal{L}(T \parallel \mathrm{Inhib}_H)$ contains a single timed word

$$(begin\_ctrl, 2)(end\_ctrl, 4) \cdots (begin\_ctrl, 2 + 4i)(end\_ctrl, 4(i + 1)) \cdots \tag{3}$$

Consider now the general behavior of the system, i.e. all actions are enabled with no restrictions. When *cloche* occurs, the system moves to state *s2*. When the system is in *s2* it is handling the *cloche* action. This operation normally requires one time unit, after which the system returns to the initial state. However, while being in *s2*, the system can catch other *cloche* actions and in this case it moves to state *s3*. If the handling of them requires too much time ($x0$ becomes greater than 2), it is necessary to postpone the beginning of the successive control cycle (i.e. action *reset* is executed).

Thus, the *cloche* actions can interfere with the basic system behavior if they are too close to each other. To see this formally with our approach to non-interference consider a natural number

n and the automaton $T \parallel \mathrm{Interf}_H^n$. If $n = 0$ high-level actions can occur without restrictions on their relative delays and consequently the *reset* action could be executed. Thus, the system does not satisfy our non-interference definition. If, instead, $n \geq 1$, then only one *cloche* action can occur between two successive control cycles and it is managed without affecting the basic behavior. Thus, for each $n \geq 1$, $(T \parallel \mathrm{Interf}_H^n)/H$ generates the single timed word (3) which in turn is the unique timed word generated by $T \parallel \mathrm{Inhib}_H$. From Definition 3.2 we conclude that high-level actions do not interfere in T with a minimum delay $n \geq 1$ (T is n-non-interfering for $n \geq 1$). On the other hand, if $n = 0$ there is interference.

5. Discussion

In order that the theory we have developed can become useful in practice, we must check the equivalence of timed automata. This implies checking inclusion in both directions of the respective recognized languages. It is well known that for general timed automata the language inclusion problem is undecidable (see [2]). However, the language inclusion problem can be solved if the system can be modeled using deterministic automata [2] or event-clock automata [4]. To make the method effective also in the general case, we are following two directions. On one side we are studying weaker notions of non-interference which are checkable on general timed automata. On the other side, we are investigating sufficient conditions on the structure of the automaton T representing the system, which allow us to decide the equivalence check required by our timed non-interference definition. This direction is promising since the automata that have to be compared have a very similar structure, derived from the initial automaton T.

References

[1] Aceto, L., Burgueño, A. and Guldstrand Larsen, K. Model Checking via Reachability Testing for Timed Automata. TACAS 98, Springer LNCS, 1384, 263–280, 1998.

[2] Alur, R. and Dill, D.L. A Theory of Timed Automata. *Theoretical Computer Science*, 126, 183–235, 1994.

[3] Alur, R., Courcoubetis, C. and Dill, D.L. Model-Checking in Dense Real-time. *Information and Computation*, 104, 2–34, 1993.

[4] Alur, R., Fix, L. and Henzinger, T.A. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theoretical Computer Science*, 204, 1997.

[5] Focardi, R. and Gorrieri, R. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1): 5–33, 1995.

[6] Focardi, R., Ghelli, A. and Gorrieri, R. Using Non Interference for the Analysis of Security Protocols. In Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols (H. Orman and C. Meadows Eds.) September 3-5, 1997, DIMACS Center, CoRE Building, Rutgers University.

[7] Focardi, R. and Gorrieri, R. Automatic Compositional Verification of Some Security Properties. In Proceedings of Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'96). Springer LNCS, 1055, 167–186, 1996.

[8] Focardi, R. and Gorrieri, R. The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. *IEEE Transactions on Software Engineering*, 23(9): 550–571, 1997.

[9] Focardi, R., Gorrieri, R. and Martinelli, F. Information Flow in a Discrete-Time Process Algebra In Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00), (P. Syverson ed.), IEEE press, Cambridge, England, July 2000.

[10] Goguen, J.A. and Meseguer, J. Security Policy and Security Models. In Proceedings of the 1982 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 11–20, 1982.

[11] Henzinger, T.A., Nicollin, X., Sifakis, J. and Yovine, S. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111, 193–244, 1994.

[12] McLean, J. A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. In Proceedings of 1994 IEEE Symposium on Research in Security and Privacy, IEEE Press, 1994.

[13] Ryan, P. Y. A. and Schneider, S. A. Process Algebra and Non-Interference. *Journal of Computer Security*, 9(12), 75–103, 2001.

[14] Roscoe, A.W., Woodcock, J.C.P. and Wulf, L. Non-Interference Through Determinism. *Journal of Computer Security*, 4(1), 1996.

[15] Sabelfeld, A. and Sands, D. Probabilistic Noninterference for Multi-threaded Programs. In Proceedings of the 13th IEEE Computer Security Foundations Workshop, Cambridge, England, July 2000. IEEE Computer Society Press, 2000.

[16] Yovine, S. Model Checking Timed Automata. Lectures on Embedded Systems, Springer LNCS, 1494, 114–152, 1996.