Centre Fédéré en Vérification

Technical Report number 2008.103

Control of Infinite Symbolic Transitions Systems under Partial Observation

Gabriel Kalyon, Tristan Le Gall, Hervé Marchand, Thierry Massart

This work was partially supported by a FRFC grant: 2.4530.02
and by the MoVES project. MoVES (P6/39) is part of the IAP-Phase VI Interuniversity Attraction Poles Programme funded by the Belgian State, Belgian Science Policy

http://www.ulb.ac.be/di/ssd/cfv

Control of Infinite Symbolic Transition Systems under Partial Observation

Gabriel Kalyon[1,⋆] , Tristan Le Gall[1], Hervé Marchand[2], and Thierry Massart[1,⋆⋆]

[1] Université Libre de Bruxelles (U.L.B.), First.Last@ulb.ac.be
[2] IRISA/INRIA, Campus de Beaulieu, Rennes, France, First.Last@irisa.fr

Abstract. We propose algorithms for the synthesis of memoryless controllers through partial observation of infinite state systems modelled by Symbolic Transition Systems. We provide models of safe controllers both for potentially blocking and non blocking controlled systems. To obtain algorithms for these problems, we use abstract interpretation techniques which provide over-approximations of the transitions set to be disabled. To our knowledge, with the hypotheses taken, the improved version of our algorithm provides a better solution than what was previously proposed in the literature. Our tool SMACS allowed us to make an empirical validation of our methods to show their feasibility and usability.

Keywords: Symbolic Transition Systems, Control Synthesis, Partial Observation, Abstract Interpretation.
AMS Classification: 93C65 Discrete event systems, 93C83 Control problems involving computers, 03C95 Abstract model theory.

1 Introduction

Discrete event systems control theory provides synthesis methods for a controller that usually has a full observation of the plant, modelled by a finite state system and can disable controllable actions. This simple and optimistic view of the problem is not always satisfactory. Indeed, in practice, the controller interacts with the plant through sensors and actuators, and an extended model with variables may be better suited to specify the plant. In that case, to provide an homogeneous treatment of these models, it is convenient to consider infinite variables domains. Moreover, the hypothesis of full observation can generally not be made either because the sensors only have finite precision or because some parts of the plant are not observed by the controller.

In this paper, we address the controller synthesis of partially observed infinite state systems to solve the *state avoidance problem*, where the controller's goal

⋆ Supported by the Belgian National Science Foundation (FNRS) under a FRIA grant.
⋆⋆ This work has been done in the MoVES project (P6/39) which is part of the IAP-Phase VI Interuniversity Attraction Poles Programme funded by the Belgian State, Belgian Science Policy.

consists in preventing the system from reaching a specified set of states *Bad*. We use Symbolic Transition Systems (STS) [9] to model the plant, where an STS is a transition system defined over a set of variables whose domain can be infinite, each transition is *guarded* on the system variables, and has an *update* function which indicates the variables changes when the transition is fired. Furthermore, transitions are labelled with symbols taken from a finite alphabet. The semantics of an STS is therefore given by a potentially infinite state labelled transition system where the states are valuations of the variables.

When control specifications are defined on the system states, it is more natural and more useful to consider a controller observing the system through its states [21]. Moreover, the controller gets in general only *partial observation*, because of the imprecision of the observing material. So, we follow the approach taken by [13], where the partial observation is modelled by a mask, corresponding to a mapping from the state space to an (infinite) observation space.

Related works The *controller synthesis of finite state systems with partial observation* of the actions has been widely studied in various works. The problem with partial observation on the states (mask) has been introduced by Kumar et al in [13]. In [20] properties of *M-controllability* give sufficient conditions to ensure controllability. To synthesize the controlled system, they use a *forward* approach with a *post* operator. Hill et al. extend this work in [10] and provide a method which synthesizes more permissive controllers, but with a different hypothesis on the masks. Since we take infinite state systems and use abstract interpretation techniques, we have preferred a *backward* approach. In game theory, the controller synthesis problem can be stated as the synthesis of a winning strategy in a two players game between the plant and the controller. The cases of *imperfect and incomplete information games* have been studied for finite state systems (see e.g. [4]).

Controller synthesis of infinite state systems modelled by STS in the case of full observation has been examined in a previous work [15]. We used abstract interpretation techniques to ensure that the controlled system can be effectively computed. We showed that, since these abstract interpretation techniques induce an over-approximation of the computations, this implies that the computed controlled system in not always the most permissive. In [14], Kumar and Garg extend their previous work [13] to consider infinite systems. They prove that, in that case, the state avoidance control problem is undecidable. They also show that the problem can be solved in the case of Petri nets, when the set *Bad* is upward-closed. The controller synthesis of *infinite state systems* modelled by Petri nets has also been considered in [11].

In order to deal with the infiniteness of state space, the algorithms presented in this paper are symbolic: they do not enumerate individual states, but deal with the system variables by means of symbolic computations and the use of predicate transformers. Moreover, since the problem is undecidable, we use abstract interpretation techniques (see e.g. [5, 8, 12]) to get effective algorithms. It is worth noticing that both concrete and abstract domains can be infinite (the abstract domain is a simpler domain, which substitutes the concrete domain

in abstract interpretation techniques). Those algorithms were implemented in a tool named SMACS.

In section 2, we introduce our model for infinite systems to control. In section 3, we define the control mechanisms we can use and we define the state avoidance control problem. In section 4, we present a semi-algorithm, which attempts to solve our problem. In section 5, we explain how to obtain an algorithm using abstract interpretation techniques. In section 6, we experimentally validate our method on various examples.

2 Symbolic Transition Systems

The (infinite) domain of a variable v is denoted $\mathcal{D}_v$. If $V = \langle v_1, \ldots, v_n \rangle$ is a tuple of variables, we note $\mathcal{D}_V = \prod_{i \in [1,n]} \mathcal{D}_{v_i}$. A *valuation* $\boldsymbol{\nu}$ of V is a tuple $\langle \boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_n \rangle \in \mathcal{D}_V$. A *predicate* over a tuple V is defined as a subset $P \subseteq \mathcal{D}_V$ (a states set for which the predicate holds). The complement of a set $H \subseteq \mathcal{D}_V$ is denoted by $\overline{H}$. The preimage function of $f : D_1 \to D_2$ is denoted by $f^{-1} : D_2 \to 2^{D_1}$.

Definition 1 (Symbolic Transition System). *A* symbolic transition system *(STS) is a tuple $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$ where:*

- *$V = \langle v_1, \ldots, v_n \rangle$ is a tuple of variables*
- *$\Theta \subseteq \mathcal{D}_V$ is a predicate on V defining the initial condition on the variables*
- *Σ is a finite alphabet of actions*
- *Δ is a finite set of symbolic transitions $\delta = \langle \sigma_\delta, G_\delta, A_\delta \rangle$ where:*
  - *$\sigma_\delta \in \Sigma$ is the action of δ,*
  - *$G_\delta \subseteq \mathcal{D}_V$ is a predicate on V guarding δ,*
  - *$A_\delta : \mathcal{D}_V \mapsto \mathcal{D}_V$ is the update function of δ.*

Given an action $\sigma \in \Sigma$, we define the set of transitions labelled by σ as $\text{Trans}(\sigma) = \{\delta \in \Delta \mid \sigma_\delta = \sigma\}$. The semantics of an STS is a possibly infinite Labelled Transition System (LTS) where states are valuations of its variables:

Definition 2 (STS's Semantics). *The semantics of an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$ is an LTS $[\![\mathcal{T}]\!] = \langle Q, Q_o, \Sigma, \to \rangle$ where:*

- *$Q = \mathcal{D}_V$ is the set of states*
- *$Q_0 = \Theta$ is the set of initial states*
- *Σ is the set of labels*
- *$\to \subseteq Q \times \Sigma \times Q$ is the transition relation defined as $\{\langle \boldsymbol{\nu}, \sigma, \boldsymbol{\nu}' \rangle \mid \exists \delta \in \Delta : (\sigma_\delta = \sigma) \wedge (\boldsymbol{\nu} \in G_\delta) \wedge (\boldsymbol{\nu}' = A_\delta(\boldsymbol{\nu}))\}$.*

Initially, an STS is in one of its initial states. A transition can only be fired if its guard is satisfied and when fired, the variables are updated according to its update function. If no transition can be fired from a state $\boldsymbol{\nu} \in \mathcal{D}_V$, i.e. $\forall \delta \in \Delta : \boldsymbol{\nu} \notin G_\delta$, we say that this state is *blocking*.

Given an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, $\mathsf{reachable}(\mathcal{T}) \subseteq \mathcal{D}_V$ is defined as the set of states that are reachable from an initial state in $[\![\mathcal{T}]\!]$.

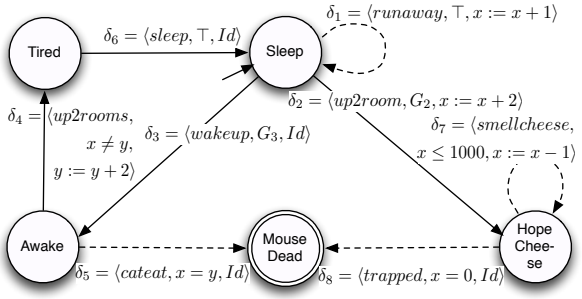

Fig. 1. The cat and mouse example

An STS may be defined with explicit locations. This is equivalent to having a finite variable of enumerated type, which encodes the locations. Therefore, in our examples, we generally represent STS using locations.

Example 1. The STS of Fig. 1 illustrates a modified version of the cat and mouse example given in [18]. *Id* denotes the identity function. Fig. 1 will be used in this paper, with different values for the guards G_2 and G_3 (initially $\top$). The STS has explicit locations ℓ and two variables: x (resp. y) identifies the room number occupied by the mouse (resp. the cat). A system state is a triple $\langle \ell, x, y \rangle$. The initial condition is given by the state $\langle Sleep, 1, 0 \rangle$. When the cat wakes up, she can eat the mouse if both are in the same room, or move and sleep again. In the location HopeCheese, if the mouse is in one of the first 1000 rooms, he can smell the cheese and moves to the room 0, where he is killed by a trap.

3 State avoidance control problem

In this section, we define the state avoidance control problem w.r.t. the kind of information and the available control mechanisms.

3.1 Means of Observation

We consider systems with partial observation, where there is an uncertainty about the real state of the system. This partial observation is formally defined by a mask $M : \mathcal{D}_V \to Y$, which corresponds to a mapping from the state space $\mathcal{D}_V$ to the (possibly infinite) observation space Y. So, Y can be seen as a partition of $\mathcal{D}_V$, where each equivalence class contains the states with the same mask.

Example 2. For the system of Fig. 1, the localization of the cat is unknown. So, the mask $M : Loc \times \mathbb{N} \times \mathbb{N} \to Loc \times \mathbb{N}$ is defined as follows: $M(\langle \ell, x, y \rangle) = \langle \ell, x \rangle$.

In the sequel, we consider three kinds of partial observation:

1. two locations (or more) give the same observation: in this case, the controller is not sure about the exact location of the system.

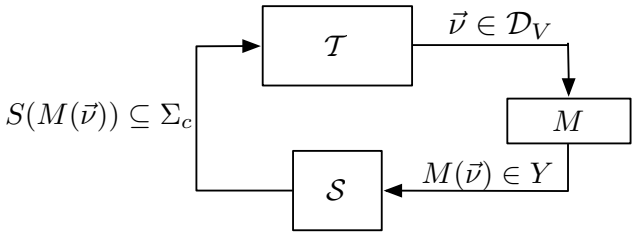

Fig. 2. Control under partial information

2. some variables are hidden: the controller cannot determine the value of those variables.
3. the value of a numerical variable is unknown if this value belongs to a specified interval. This mask implements variables that are *partially hidden*.

3.2 Means of Control

The control mechanism is similar to the one defined in [19, 3] : the alphabet $\Sigma = \Sigma_c \cup \Sigma_{uc}$ is partitioned into Σ_c, the set of controllable actions, and Σ_{uc}, the set of uncontrollable ones. As a consequence, the set Δ is partitioned accordingly to Δ_c and Δ_{uc}.

3.3 Controller and controlled system

The controller aims to restrict the system's behavior and to prevent it from reaching some bad states. The controller with partial observation (see Fig. 2) is formally defined as follows:

Definition 3 (Controller). *Given an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, and a mask $M : \mathcal{D}_V \mapsto Y$, a controller for $\mathcal{T}$ is a pair $\mathcal{C} = \langle \mathcal{S}, E \rangle$, where:*

- *$\mathcal{S} : Y \to 2^{\Sigma_c}$ is a supervisory function which defines, for an observation $y \in Y$, a set $\mathcal{S}(y)$ of controllable actions to forbid in any state $\boldsymbol{\nu}$ such that $y = M(\boldsymbol{\nu})$*
- *$E \subseteq \mathcal{D}_V$ is a set of states to forbid, which restricts the set of initial states.*

The behavior of the controlled system is defined as follows:

Definition 4 (Controlled STS). *Given an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, a mask $M : \mathcal{D}_V \mapsto Y$, and a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, the system $\mathcal{T}$ controlled by $\mathcal{C}$, is an STS $\mathcal{T}_{/\mathcal{C}} = \langle V, \Theta_{/\mathcal{C}}, \Sigma, \Delta_{/\mathcal{C}} \rangle$, where:*

- *$\Theta_{/\mathcal{C}} = \Theta \setminus E$*
- *$\Delta_{/\mathcal{C}}$ is defined using the following rule:*

$$\frac{\langle \sigma, G, A \rangle \in \Delta \quad G_{/\mathcal{C}} = G \setminus \{\boldsymbol{\nu} \in \mathcal{D}_V | \sigma \in \mathcal{S}(M(\boldsymbol{\nu}))\}}{\langle \sigma, G_{/\mathcal{C}}, A \rangle \in \Delta_{/\mathcal{C}}}$$

The supervisory function $\mathcal{S}$ allows us to restrict the guards of the controlled system. Indeed, a transition δ can no longer be fired in $\mathcal{T}_{/\mathcal{C}}$ from a state $\boldsymbol{\nu}$, if its action $\sigma_\delta \in \mathcal{S}(M(\boldsymbol{\nu}))$. This function satisfies the *S-observability* condition meaning that if $\boldsymbol{\nu}$ and $\boldsymbol{\nu}'$ have the same observation, then $\mathcal{S}$ will have the same control decision for both states.

3.4 Definition of the problems

We focus on two variants of the state avoidance control problem :

Problem 1 (Basic state avoidance control problem) *For an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, a mask $M : \mathcal{D}_V \mapsto Y$ and a predicate Bad, i.e. a set of forbidden states, the basic state avoidance control problem consists in building a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that* $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}) \cap Bad = \varnothing$.

A solution to this first problem does not ensure that the controlled system is dead-lock free (i.e. it is not ensured that the controlled system has always the possibility to make a move). To ensure this important property, we define a second problem. Note that in this paper, we use the term non-blocking instead of dead-lock free (generally, in the literature the non-blocking property means that the controller has to ensure the reachability of some states).

Problem 2 (Non-blocking state avoidance control problem) *This problem consists in defining a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that (i)* $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}) \cap Bad = \varnothing$, *and (ii)* $\forall \boldsymbol{\nu} \in \mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}), \exists \delta \in \Delta_{/\mathcal{C}} : \boldsymbol{\nu} \in (G_{/\mathcal{C}})_\delta$.

We can immediately notice that a trivially correct controller (for both problems) is one where $E = \mathcal{D}_V$. Therefore, the notion of permissiveness has been introduced to compare the quality of different controllers for a given STS.

Definition 5 (Permissiveness[3]). *Given an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, and a mask $M : \mathcal{D}_V \mapsto Y$, a controller $\mathcal{C}_1 = \langle \mathcal{S}_1, E_1 \rangle$ is more permissive than a controller $\mathcal{C}_2 = \langle \mathcal{S}_2, E_2 \rangle$, iff* $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_1}) \supseteq \mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_2})$.

Unfortunately, we cannot get a optimal result in terms of permissiveness.

Proposition 1. *As solution of Problem 1 or 2, in general, no controller is the most permissive.*

Proof. To prove this property, we consider the following example

For the LTS of Fig. 3, the set of initial states $Q_0 = \{x_1, x_2\}$ and all transitions are controllable. The set $Bad = \{x_5, x_6\}$ and the mask M is defined as follows:

$$M(x) = \begin{cases} y_1 \text{ if } x \in \{x_1, x_4\} \\ y_2 \text{ if } x \in \{x_2, x_3\} \\ y_3 \text{ if } x \in \{x_5, x_6\} \end{cases}$$

There are three possibilities to avoid the set Bad:

[3] Other kinds of permissiveness like language or execution can be conbsidered to compare the quality of controllers

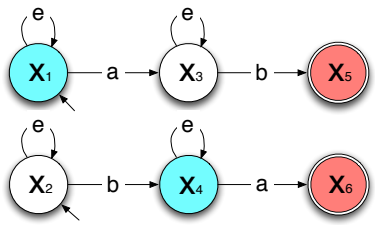

Fig. 3. System without a most permissive controller

- to forbid the transition a in the observation state y_1: $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_1}) = \{x_1, x_2, x_4\}$.
- to forbid the transition b in the observation state y_2: $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_2}) = \{x_1, x_2, x_3\}$.
- to forbid the transitions a and b everywhere: $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_3}) = \{x_1, x_2\}$.

$\mathcal{C}_1$ and $\mathcal{C}_2$ are both more permissive than $\mathcal{C}_3$ but are not comparable. Thus, there does not exist a most permissive controller. □

Proposition 2. *If we restrict the problem in finding a controller $\mathcal{C}$ such that no more permissive controller $\mathcal{C}'$ exists, the basic and non-blocking state avoidance control problems are undecidable.*

Proof. Under full observation, the computation of the maximally permissive controller solving the state avoidance control problem is undecidable [14]. The restriction of this problem to the basic state avoidance control problem is trivial using the identity function as mask and gives the most permissive controller.

The restriction of the restricted non-blocking state avoidance control problem to the restricted basic state avoidance control problem is also trivial. □

Hence, our aim is to find correct controllers that, are permissive enough to be of good practical value. Our experiments will validate our solutions.

4 Symbolic Computation of the controller (An exact Computation)

We present a theoretical framework to synthesize a controller which attempts to solve Problem 1; we then extend this result to the non-blocking case. From Proposition 2, it is clear that this framework, where no approximation is done, can only provide semi-algorithms.

The general idea of the control is to compute, using fixpoint computation, the set $I(Bad)$ of states that can lead to Bad triggering only uncontrollable transitions or that can be blocking after control (for the non-blocking case). Then, based on this set of states, we compute the controller, whose aim is to disable, for each observation $y \in Y$, all the controllable actions that may lead

to a state in $I(Bad)$. Our algorithms are symbolic in the sense that they do not enumerate the state space; the feasibility of their computations is discussed in section 5.

4.1 The basic state avoidance control problem

We describe here a symbolic method to compute a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ that solves Problem 1.

Computation of $I(Bad)$ This set of states and more generally $I(.)$ is given by the function $\text{Coreach}_{uc} : 2^{\mathcal{D}_V} \to 2^{\mathcal{D}_V}$ defined below. This set corresponds to the set of states that lead to Bad firing only uncontrollable transitions.

Classically, we first define the function $\text{Pre}_{uc}(B)$, which computes the set of states from which a state of B is reachable by triggering exactly one uncontrollable transition.

$$\text{Pre}_{uc}(B) = \bigcup_{\delta \in \Delta_{uc}} \text{Pre}(\delta, B) \text{ , where} \tag{1}$$

$$\text{Pre}(\delta, B) = G_\delta \cap A_\delta^{-1}(B) \tag{2}$$

We recall that G_δ is the set of states from which δ can be fired and $A_\delta^{-1}(B)$ is the set of states that lead to B by δ.

Further, $\text{Coreach}_{uc}(Bad)$ is obtained by computing the following fixpoint equation:

$$\text{Coreach}_{uc}(Bad) = \text{lfp}(\lambda B. Bad \cup \text{Pre}_{uc}(B)) \tag{3}$$

Note that the limit of the fixpoint $\text{Coreach}_{uc}(Bad)$ actually exists as the function Coreach_{uc} is monotonic (but may be uncomputable).

Computation of the controller $\mathcal{C}$ and of the controlled system $\mathcal{T}_{/\mathcal{C}}$ We first define a function $\mathcal{F} : \Sigma \times 2^{\mathcal{D}_V} \to 2^Y$: for an action $\sigma \in \Sigma$ and a set $B \subseteq \mathcal{D}_V$ of states to forbid, $\mathcal{F}(\sigma, B)$ specifies the set of observation states for which the action σ has to be forbidden, i.e. the set of observations $y \in Y$ such that there exists $\boldsymbol{\nu} \in \mathcal{D}_V$ with $M(\boldsymbol{\nu}) = y$, from which a transition labelled by σ leads to B.

$$\mathcal{F}(\sigma, B) = \begin{cases} \bigcup_{\delta \in \text{Trans}(\sigma)} M(\text{Pre}(\delta, B) \setminus B) & \text{if } \sigma \in \Sigma_c \\ \varnothing & \text{otherwise} \end{cases} \tag{4}$$

The controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ is defined as follows:

- the supervisory function $\mathcal{S}$ is:

$$\forall y \in Y, \mathcal{S}(y) = \{\sigma \in \Sigma \mid y \in \mathcal{F}(\sigma, I(Bad))\} \tag{5}$$

- the set E is:

$$E = I(Bad) \tag{6}$$

The controlled system $\mathcal{T}_{/\mathcal{C}}$ is computed using definition 4 with the system $\mathcal{T}$ and the controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ defined as above.

Proposition 3. *Given a system $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, a mask $M : \mathcal{D}_V \to Y$ and a predicate Bad, i.e. a set of forbidden states, the controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, where $\mathcal{S}$ and E are computed at (5) and (6), solves Problem 1.*

Proof. We prove by induction on the length n of the executions that $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}) \cap I(Bad) = \varnothing$. This implies that $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}) \cap Bad = \varnothing$.

- Base ($n = 0$): the initial states of the controlled system $\mathcal{T}_{/\mathcal{C}}$ are defined by $\Theta_{/\mathcal{C}} = \Theta \setminus E = \Theta \setminus I(Bad)$. Thus, the execution of $\mathcal{T}_{/\mathcal{C}}$ starts in a state that does not belong to $I(Bad)$.
- Induction: suppose the proposition holds for paths of transitions of length less or equal to n. We show that no transition $\delta \in \Delta$ can be fired from a state $\boldsymbol{\nu} \notin I(Bad)$ to a state $\boldsymbol{\nu}' \in I(Bad)$. Two cases are *a priori* possible. Both are *a posteriori* impossible:
  - either $\delta \in \Delta_c$, then this transition cannot be fired since $\sigma_\delta \in \mathcal{S}(M(\boldsymbol{\nu}))$ by (4) and (5).
  - or $\delta \in \Delta_{uc}$, then $\boldsymbol{\nu} \in I(Bad)$, which is impossible by hypothesis.

□

Example 3. For the STS of Fig. 1 and the mask of Example 2, we define Bad as $\{\langle MouseDead, k_1, k_2 \rangle | k_1, k_2 \in \mathbb{N}\}$. The controllable (resp. uncontrollable) transitions are those drawn in plain (resp. dashed) lines. Then, $I(Bad) = \{\langle HopeCheese, k_1, k_2 \rangle | k_1 \in [0, 1000] \wedge k_2 \in \mathbb{N}\} \cup \{\langle Awake, k_1, k_1 \rangle | k_1 \in \mathbb{N}\} \cup \{\langle MouseDead, k_1, k_2 \rangle | k_1, k_2 \in \mathbb{N}\}$. The computation of $\mathcal{F}$ gives:

$$\mathcal{F}(\sigma, I(Bad)) = \begin{cases} \{\langle Sleep, k_1 \rangle | k_1 \in \mathbb{N}\} & \text{if } \sigma = wakeup \\ \{\langle Sleep, k_1 \rangle | k_1 \in [0, 998]\} & \text{if } \sigma = up2rooms \\ \varnothing & \text{otherwise} \end{cases}$$

Then, the supervisory function $\mathcal{S}$ is defined as follows:

$$\mathcal{S}(y) = \begin{cases} \{wakeup, up2rooms\} & \text{if } y \in \{\langle Sleep, k_1 \rangle | k_1 \in [0, 998]\} \\ \{wakeup\} & \text{if } y \in \{\langle Sleep, k_1 \rangle | k_1 \geq 999\} \\ \varnothing & \text{otherwise} \end{cases}$$

The controlled system is given by Fig. 1, with the guards $G_2 = (x \geq 999)$ and $G_3 = \bot$. ◇

4.2 The non-blocking state avoidance control problem

We describe here a symbolic method to compute a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ that solves Problem 2.

Computation of $I(Bad)$ (first method) This set of states and more generally $I(.)$ is given by the function $\text{Coreach}_{uc}^{nb} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ defined below. This set corresponds to the set of states that would be blocking in the controlled system and of states that lead to a forbidden state firing only uncontrollable transitions.

To compute $\text{Coreach}_{uc}^{nb}(Bad)$, we first compute $\text{Coreach}_{uc}(Bad)$ (defined by (3)). Then, if we make unreachable the forbidden states by cutting all the controllable transitions that lead to a bad state, the corresponding controlled system $\mathcal{T}_{/\mathcal{C}}$ could have new blocking states. We must add these blocking states to the set of forbidden states. The function $\text{Pre}_{bl}(B)$ computes, for a set $B \subseteq \mathcal{D}_V$ of states to forbid, the set of states, that would be blocking in the controlled system, if the states of B were no longer reachable. The computation of the blocking states is based on the function $\mathcal{F}$ defined at (4). To ensure the convergence in the computation of $\text{Coreach}_{uc}^{nb}(Bad)$, Pre_{bl}, and therefore $\mathcal{F}$, must be monotonic. Thus, we use the monotonic function $\hat{\mathcal{F}}$ instead of $\mathcal{F}$ in the computation of the controller for the non-blocking case.

$$\hat{\mathcal{F}}(\sigma, B) = \begin{cases} \bigcup_{\delta \in \text{Trans}(\sigma)} M(\text{Pre}(\delta, B)) & \text{if } \sigma \in \Sigma_c \\ \varnothing & \text{otherwise} \end{cases}$$

Remark 1. Note that $\hat{\mathcal{F}}$ is more restrictive than $\mathcal{F}$ and thus a controller computed w.r.t. $\mathcal{F}$ is more permissive than a controller computed w.r.t. $\hat{\mathcal{F}}$.

We now explain how to compute the blocking states in the controlled system $\mathcal{T}_{/\mathcal{C}}$. A state $\boldsymbol{\nu} \in \mathcal{D}_V$ is blocking in $\mathcal{T}_{/\mathcal{C}}$, if the two following conditions are satisfied in the system $\mathcal{T}$:

1. the state $\boldsymbol{\nu}$ has no outgoing uncontrollable transition.
2. every outgoing controllable transition δ of $\boldsymbol{\nu}$ is forbidden by control in the observation state $M(\boldsymbol{\nu})$, i.e. $M(\boldsymbol{\nu}) \in \hat{\mathcal{F}}(\sigma_\delta, B)$

Proposition 4. *Formally, a state $\boldsymbol{\nu}$ is blocking whenever:*

1. $\forall \delta \in \Delta_{uc} : \boldsymbol{\nu} \notin G_\delta$
2. $\forall \delta \in \Delta_c : (\boldsymbol{\nu} \notin G_\delta) \vee (M(\boldsymbol{\nu}) \in \hat{\mathcal{F}}(\sigma_\delta, B))$

Because $\hat{\mathcal{F}}(\sigma, B)) = \varnothing$ $(\forall \sigma \in \Sigma_{uc})$, the function Pre_{bl}, which computes the states that would be blocking in the controlled system, can be expressed as follows:

$$\text{Pre}_{bl}(B) = B \cup \left[\bigcap_{\delta \in \Delta} \left(\overline{G_\delta} \cup (M^{-1}(\hat{\mathcal{F}}(\sigma_\delta, B))) \right) \right]$$

Adding the blocking states to the forbidden states can provide new states leading uncontrollably to a forbidden state. Consequently, to compute the set $\text{Coreach}_{uc}^{nb}(Bad)$, we define the following fixpoint equation:

$$\text{Coreach}_{uc}^{nb}(Bad) = \text{lfp}(\lambda B.Bad \cup \text{Pre}_{bl}(\text{Coreach}_{uc}(B))) \tag{7}$$

The controller and the controlled system are defined similarly to what is done at the point 4.1.

Proposition 5. *Given a system $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, a mask $M : \mathcal{D}_V \to Y$ and a predicate Bad, i.e. a set of forbidden states, the controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, computed according to definition 3 w.r.t. (7) solves Problem 2.*

Proof. Since $\text{Coreach}_{uc}(Bad) \subseteq \text{Coreach}_{uc}^{nb}(Bad)$, it can be proved in a similar way as the proof of Prop. 3 that Bad is not reachable in this more restrictive controlled system.

Let us suppose that the controlled system does not satisfy the non-blocking property. Then, there exists at least a blocking state $\boldsymbol{\nu} \in \mathcal{D}_V$, which is reachable in the controlled system. By definition of the fixpoint, $\boldsymbol{\nu} \in \text{Coreach}_{uc}^{nb}(Bad)$, and so is any state $\boldsymbol{\nu'} \in \mathcal{D}_V$ such that there is a sequence of uncontrollable transitions from $\boldsymbol{\nu'}$ to $\boldsymbol{\nu}$. According to definition 3, $\boldsymbol{\nu}$ and $\boldsymbol{\nu'}$ are both non reachable.

□

Computation of $I(Bad)$ (second method). This set of states and more generally $I(.)$ is given by the function $\text{Coreach}_{uc}^{nb'} : 2^{\mathcal{D}_V} \to 2^{\mathcal{D}_V}$ defined below.

$$\text{Coreach}_{uc}^{nb'}(Bad) = \text{lfp}(\lambda B.Bad \cup \text{Pre}_{bl}(\text{Pre}_{uc}(B))) \tag{8}$$

This equation computes first the set of states that lead to a forbidden state by firing only an uncontrollable transition, and then it computes the blocking states. The processing continues until the set is stabilized.

The controller can be computed as for the first method.

Proposition 6 (Avoidance states). *Given a system $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, a partial observation mask $M : \mathcal{D}_V \to Y$ and a predicate Bad, i.e. a set of forbidden states, the controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, computed according to definition 4 w.r.t. (8) solves Problem. 2.*

Proof. The proof is similar to the one of Proposition 5.

□

Proposition 7. *If the computation of the fixpoint equations at (7) and (8) finishes, then the set defined at (7) (noted $I_1(Bad)$ in the proof that follows) is equal to the set defined at (8) (noted $I_2(Bad)$ in the proof that follows).*

Proof. We first prove that $I_2(Bad) \subseteq I_1(Bad)$ as follows:

$Bad \subseteq Bad$
$\Leftrightarrow \text{Pre}_{uc}(Bad) \subseteq \text{Coreach}_{uc}(Bad)$, by (3)
$\Leftrightarrow \text{Pre}_{bl}(\text{Pre}_{uc}(Bad)) \subseteq \text{Pre}_{bl}(\text{Coreach}_{uc}(Bad))$, because Pre_{bl} is monotonic
$\Leftrightarrow (\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))^2 \subseteq (\text{Pre}_{bl}(\text{Coreach}_{uc}(Bad)))^2$, by repeating the two preceding steps
$\Leftrightarrow (\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))^{max\{k,k'\}} \subseteq (\text{Pre}_{bl}(\text{Coreach}_{uc}(Bad)))^{max\{k,k'\}}$

We then prove that $I_1(Bad) \subseteq I_2(Bad)$ as follows:

$\text{Pre}_{uc}(Bad) \subseteq \text{Pre}_{uc}(Bad)$
$\Leftrightarrow \text{Pre}_{uc}(Bad) \subseteq \text{Pre}_{bl}(\text{Pre}_{uc}(Bad))$, because $\forall X \subseteq \mathcal{D}_V : X \subseteq \text{Pre}_{bl}(X)$
$\Leftrightarrow (\text{Pre}_{uc}(Bad))^2 \subseteq \text{Pre}_{uc}(\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))$, because Pre_{uc} is monotonic
$\Leftrightarrow (\text{Pre}_{uc}(Bad))^2 \subseteq (\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))^2$, because $\forall X \subseteq \mathcal{D}_V : X \subseteq \text{Pre}_{bl}(X)$
$\Leftrightarrow (\text{Pre}_{uc}(Bad))^i \subseteq (\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))^i$, $\forall i \in \mathbb{N}$
$\Leftrightarrow (\text{Pre}_{uc}(Bad))^i \subseteq \text{Pre}_{bl}[(\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))^i]$, because $\forall X \subseteq \mathcal{D}_V$: $X \subseteq \text{Pre}_{bl}(X)$
$\Leftrightarrow \text{Pre}_{bl}[(\text{Pre}_{uc}(Bad))^i] \subseteq (\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))^{i+1}$, because Pre_{bl} is monotonic
$\Leftrightarrow \text{Pre}_{bl}(\text{Coreach}_{uc}(Bad)) \subseteq (\text{Pre}_{bl}(\text{Pre}_{uc}(Bad)))^{k''+1}$, where k" is the number of iterations to compute the fixpoint $\text{Coreach}_{uc}(Bad)$
$\Leftrightarrow [\text{Pre}_{bl}(\text{Coreach}_{uc}(Bad))]^{max\{k,k'\}} \subseteq [\text{Pre}_{bl}(\text{Pre}_{uc}(Bad))]^{max\{k,k'\}.(k_1+1)}$, by repeating the preceding steps. k_1 is the maximum between k'' and the number of iterations to compute the fixpoint $\text{Coreach}_{uc}([\text{Pre}_{bl}(\text{Coreach}_{uc}(Bad))]^i), \forall i \in [1, max\{k, k'\} - 1]$.

□

4.3 Improvement of the control algorithm

In [20], the authors define a controller which, to our knowledge, is the most permissive controller satisfying the S-observability condition known in the literature for finite systems; However, this algorithm is only defined for finite LTS. We prove the following property.

Proposition 8. *For finite systems, our algorithm solving Problem 1 gives a controller that is at least as permissive as the one obtained in [20].*

Proof. When we refer to the algorithm in [20], we use their notations.

In [20], the system to control is modelled by a finite LTS $G = \langle X, x_0, \Sigma, \delta \rangle$, where X is the set of states, x_0 is the initial state, Σ is the set of actions and $\delta : \Sigma \times X \to X$ is the transition relation. The control specification is given by a set Q of allowable states, i.e. $Q = \overline{Bad}$. The partial observation is formalized by a mask $M : X \to Y$, where Y is the finite observation space. The algorithm is composed of two steps:

1. to compute $Q^{\uparrow} \subseteq Q$. The set $Q^{\uparrow}$ is defined by $\bigcap_{j=0}^{\infty} Q_j$, where Q_j is recursively defined as follows:

$$Q_j = \begin{cases} Q & \text{if } j = 0 \\ Q \cap (\bigcap_{\sigma \in \Sigma_{uc}} \{x \in X | ((\sigma, x) \in \delta) \Rightarrow \delta(\sigma, x) \in Q_{j-1}\}) & \text{otherwise} \end{cases}$$

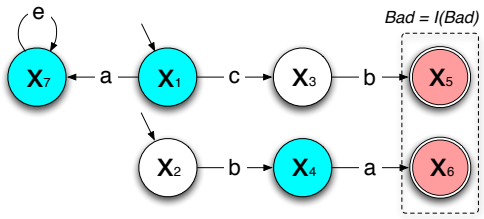

Fig. 4. Improvement of the control algorithm

2. to compute the function A, where $\forall y \in Y : A(Q^{\uparrow}, y) = \{\sigma \in \Sigma_c | \exists x \in Q^{\uparrow} : M(x) = y \wedge ((\sigma, x) \in \delta) \wedge (\delta(\sigma, x) \notin Q^{\uparrow})\}$. The forbidden actions in a state $x \in X$ are given by $A(Q^{\uparrow}, M(x))$.

We remark that $Q^{\uparrow} = \overline{\text{Coreach}_{uc}(Bad)}$, because $\forall j \geq 0, \bigcap_{i \leq j} Q_i = \overline{\bigcap_{i \leq j} \text{Pre}^i_{uc}(Bad)}$, where $\text{Pre}^0_{uc}(Bad)$ denotes Bad.

Moreover, to prevent from reaching $\text{Coreach}_{uc}(Bad)$, the function $\mathcal{S}$ is defined by $\mathcal{S}(y) = \{\sigma \in \Sigma_c | \exists x \notin \text{Coreach}_{uc}(Bad), \exists x' \in \text{Coreach}_{uc}(Bad) : M(x) = y \wedge ((x, \sigma, x') \in \rightarrow)\}$. Thus, $A(Q^{\uparrow}, y) = \mathcal{S}(y)$, $\forall y \in Y$.

□

Let us now explain how to improve it, based on the observations got from the following example.

Example 4. For the LTS of Fig. 4, the set of initial states $X_0 = \{x_1, x_2\}$ and all transitions are controllable. The set $Bad = \{x_5, x_6\}$ and the mask M is defined as follows: $M(x) = y_1$, $\forall x \in \{x_1, x_4, x_7\}$, $M(x) = y_2$, $\forall x \in \{x_2, x_3\}$ and $M(x) = y_3$, $\forall x \in \{x_5, x_6\}$.

Our algorithm forbids the transition b in the observation state $M(x_3)$ and the transition a in the observation state $M(x_4)$. However, it is sufficient to forbid b in $M(x_3)$ which makes state x_4 no longer reachable.

Based on this remark, we give an improved algorithm to compute a controller solving Problem 1 for finite systems.

Algorithm 1: Improved algorithm for finite systems

data : An STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$ such that $[\![\mathcal{T}]\!]$ is finite, a set of states $I(Bad)$ and a mask $M : X \to Y$.
returns: A controller $\mathcal{C}$ such that $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}) \cap I(Bad) = \varnothing$.

```
1 begin
2     ∀y ∈ Y, S(y) ← ∅ and C ← ⟨S, I(Bad)⟩
3     while reachable(T_/C) ∩ I(Bad) ≠ ∅ do
4         Let ν ∈ ((Pre_c(I(Bad)) \ I(Bad)) ∩ reachable(T_/C)) and δ ∈ Δ_c such
          that (ν ∈ G_δ) ∧ (A_δ(ν) ∈ I(Bad))
5         S(M(x)) ← S(M(x)) ∪ {σ_δ}
6         C ← ⟨S, I(Bad)⟩
7     return (C)
8 end
```

where $\mathrm{Pre}_c(B) = \bigcup_{\delta \in \Delta_c} \mathrm{Pre}(\delta, B)$, for $B \subseteq \mathcal{D}_V$.

The idea of this algorithm is to choose a state $\boldsymbol{\nu} \notin I(Bad)$, which is reachable in the current controlled system, and a transition δ leading to $I(Bad)$ from $\boldsymbol{\nu}$, and to forbid σ_δ in the observation state $M(\boldsymbol{\nu})$. This operation is repeated until the set $I(Bad)$ is no longer reachable in the current controlled system. So, the main difference with the algorithm of section 4 is that we are more precise when we forbid actions. Indeed, we verify that a state is still reachable in the current controlled system, before deciding to forbid an action in the corresponding observation state.

Algorithm 1 solves Problem 1 and it always outperforms or gives the same result than the one defined in section 4 (and thus the one in [20]).

4.3.1 Adaptation for infinite systems The algorithm of the preceding section may not terminate for infinite systems. So, we define an adapted version of this algorithm, which is less better but which terminates.

Algorithm 2: Improved algorithm for infinite systems

data : STS $\mathcal{T} = \langle \mathcal{D}_V, \Theta, \Sigma, \Delta \rangle$, a set of states $I(Bad)$ and a mask $M : X \to Y$.
returns: A controller $\mathcal{C}$ such that $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}) \cap I(Bad) = \varnothing$.

```
1 begin
2     forall σ ∈ Σ do
3         Compute F(σ, I(Bad))
4         Compute S from F
5         C = ⟨S, I(Bad)⟩
6         if reachable(T_/C) ∩ I(Bad) = ∅ then
7             break
8     return (C)
9 end
```

The algorithm 1 does not terminate for infinite systems, because we enumerate a possibly infinite set of states (see line 4). To overcome this problem, we forbid an action σ for all the states for which it is necessary (i.e. computation of $\mathcal{F}(\sigma, I(Bad))$). And we repeat this operation until the set $I(Bad)$ is no more reachable in the current controlled system. This algorithm is at most even better as the algorithm 1 and it is at least even better as the one of the section 4. And this algorithm solves the problem 1.

4.4 Improvement of the control algorithm using memory

To improve the controlled system's permissiveness, we can use a controller with memory. In fact, we consider an 1-order memory for the controller, which allows to retain the possible current states of $\mathcal{T}$. When the controller gets the observation y_i from the system, it computes the possible current states P_i at the step i according to the possible current states P_{i-1} at the step $i-1$ (which is retained in the memory) and the control action at the step $i-1$. Then, the control action is computed to prevent from reaching $I(Bad)$ from the possible current states P_i. Formally:

- initial step: $P_1 = M^{-1}(y_1) \cap \Theta$ and $\mathcal{S}$ is defined by:

$$\forall \sigma \in \Sigma_c : (\sigma \in \mathcal{S}(P_1) \Leftrightarrow \mathrm{Post}_\sigma(P_1) \cap I(Bad) \neq \varnothing) \tag{9}$$

- inductive step ($\forall i > 0$): $P_i = M^{-1}(y_i) \cap \mathrm{Post}_{\mathcal{S}}(P_{i-1})$ and $\mathcal{S}$ is defined by:

$$\forall \sigma \in \Sigma_c : (\sigma \in \mathcal{S}(P_i) \Leftrightarrow \mathrm{Post}_\sigma(P_i) \cap I(Bad) \neq \varnothing) \tag{10}$$

where $\mathrm{Post}_{\mathcal{S}}(B) = \bigcup_{\sigma \in \Sigma} \bigcup_{\delta \in \mathrm{Trans}(\sigma)} \mathrm{Post}_{\mathcal{S}}(\delta, B)$ with:

$$\mathrm{Post}^{\mathcal{S}}(\delta, B) = \begin{cases} \mathrm{Post}(\delta, B) & \text{if } \sigma_\delta \notin \mathcal{S}(B) \\ \varnothing & \text{otherwise} \end{cases}$$

Note that this algorithm solves only Problem 1.

5 Effective Computation by Means of Abstract Interpretation

As seen in the previous section, the effective computation of the controller, which is based on a fixpoint equation to compute $I(Bad)$, is generally not possible for undecidability (or complexity) reasons. To overcome the undecidability problem, we use *abstract interpretation* techniques (see e.g. [5, 8, 12]), to compute an over-approximation of the fixpoint $I(Bad)$. This over-approximation ensures that the forbidden states Bad are not reachable in the controlled system, but at the price of forbidding more states than needed. Thus, we obtain a valid controller, but a stricter one.

5.1 Outline of the abstract interpretation techniques

Abstract interpretation gives a theoretical framework to the approximate solving of fixpoint equations of the form $c = F(c)$, for $c \in 2^{\mathcal{D}_V}$, where $\mathcal{D}_V$ is the state space of the system, $2^{\mathcal{D}_V}$ is a complete lattice of sets of states ordered by inclusion and F is a monotonic function. We want to compute the least fixpoint (lfp) of a monotonic function $F : 2^{\mathcal{D}_V} \to 2^{\mathcal{D}_V}$. Since $2^{\mathcal{D}_V}$ is a complete lattice, Tarski's theorem sentences that $\mathrm{lfp}(F) = \cap\{c \in 2^{\mathcal{D}_V} \mid c \supseteq F(c)\}$. So, any post fixpoint is an over-approximation of $\mathrm{lfp}(F)$.

Our aim is to compute a post fixpoint of F, according to the following method:

1. the concrete domain, *i.e.* the sets of states $2^{\mathcal{D}_V}$ is substituted by a simpler abstract domain Λ (static approximation), both domains having lattice structure. The concrete lattice $(2^{\mathcal{D}_V},\subseteq,\cup,\cap,\varnothing,\mathcal{D}_V)$ and the abstract lattice $(\Lambda,\sqsubseteq,\sqcup,\sqcap,\bot,\top)$ are linked by a Galois connection $2^{\mathcal{D}_V} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$, which ensures the correctness of the method [5].
2. the fixpoint equation is transposed into the abstract domain. So, the equation to solve has the form: $l = F^\sharp(l)$, with $l \in \Lambda$ and $F^\sharp \sqsupseteq \alpha \circ F \circ \gamma$
3. a widening operator ∇ (dynamic approximation) ensures that the fixpoint computation converges after a finite number of steps to some upper-approximation l_∞.
4. the concretization $c_\infty = \gamma(l_\infty)$ is an over-approximation of the least fixpoint of the function F.

For our experiments, we chose the abstract lattice of *convex polyhedra* [6]. A convex polyhedron on the tuple of variables $\langle v_1, \ldots, v_n \rangle$ is defined as a conjunction of k linear constraints; for example, $v_1 \geq 0 \wedge v_2 \geq 0 \wedge v_1 + v_2 \leq 1$ defines a square triangle.

In this lattice, $\sqcap$ is the classical intersection, $\sqcup$ is the convex hull and $\sqsubseteq$ is the inclusion. The widening operator [6] $P_1 \nabla P_2$ roughly consists in removing from P_1 all the constraints not satisfied by P_2. In other words, its principle is: if the value of a variable or a linear expression grows between two steps of the fixpoint computation, then one guesses that it can grow indefinitely.

We assume in the sequel that the abstract lattice $\Lambda(\sqsubseteq, \sqcup, \sqcap, \top, \bot)$, the functions $\alpha : \wp(S) \to \Lambda$, $\gamma : \Lambda \to \wp(S)$ and the widening operator $\nabla : \Lambda \to \Lambda$ are defined, with $\wp(S) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$.

5.2 Computation of the controlled system using abstract interpretation

The function corresponding to $\mathrm{Pre}_{uc} : 2^{\mathcal{D}_V} \mapsto 2^{\mathcal{D}_V}$ is named $\mathrm{Pre}^\sharp_{uc} : \Lambda \mapsto \Lambda$, and is defined in the following way. For $l \in \Lambda$, we have:

$$\mathrm{Pre}^\sharp_{uc}(l) = \bigsqcup_{\delta \in \Delta_{uc}} \mathrm{Pre}^\sharp(\delta, l) \text{ , where} \tag{11}$$

$$\mathrm{Pre}^\sharp(\delta, l) = \alpha(G_\delta \cap A_\delta^{-1}(\gamma(l))) \tag{12}$$

$\mathrm{Coreach}^\sharp_{uc}(Bad)$ is the least fixpoint of the function $\lambda l.\alpha(Bad) \sqcup \mathrm{Pre}^\sharp_{uc}(l)$ and we compute l_∞, defined as the limit of the sequence defined by $l_1 = \alpha(Bad)$ and $l_{i+1} = l_i \nabla \mathrm{Pre}^\sharp_{uc}(l_i)$. The abstract interpretation theory ensures that this sequence stabilizes after a finite number of steps, and that $\gamma(l_\infty)$ is an over-approximation of $I(Bad)$. So we obtain $I'(Bad) = \gamma(l_\infty)$. Finally, we define the controller as in section 4.1, using $I'(Bad)$ instead of $I(Bad)$. We do not detail the effective computation of the other fixpoint, since the same kind of transformations are involved.

Quality of the approximations The method presented here always computes a safe controller, but without any guarantee that this controller is the most permissive one. The less approximation we make during the computation, the more precise approximation of $I(Bad)$ we obtain. Even if a better approximation $I(Bad)$ does not always mean we get a better controller, generally it is the case. There are classical techniques to improve the quality of the approximations:

- the choice of the abstract lattice is the main issue: if it is not adapted to the kind of guards or assignments of the STS, the over-approximations are too rough. The practice shows that if the guards are linear constraints, and if the assignments functions are also linear, the lattice of convex polyhedra [6] works quite well.
- the computation of the fixpoint with the widening operator may be improved by several means: we can use a widening "up to" instead of the standard widening operator [8], we can use one of the fixpoint computation strategies defined in [2] and we can refine our abstract lattice (See [12] for more details).

There are however few theoretical results on the quality of the abstraction. We can only show, on some examples, that our abstractions enable the computation of a useful controller.

6 Implementation and experiments

We implemented the algorithms of sections 4 and 5. Our tool, named SMACS (Symbolic MAsked Controller Synthesis), is written in Objective CAML [17], uses the APRON library [1] and a generic fixpoint solver [7].

6.1 Description of SMACS

Variables and control structure. Unlike the model of definition 1, SMACS considers STS with explicit locations. There are two types of variables: integer or real. Events are declared controllable or uncontrollable.

Note that this model of STS allows the user to encode any tuple of variables of finite domain as locations. In particular, after a transformation of the model, we can deal with boolean variables.

Guards and assignments. The assignments are given by linear expressions; the guards are boolean combinations (and, or, not) of linear constraints. The APRON library implements several numerical abstract lattices as: intervals [5], octagons [16] and convex polyhedra [6]. Those abstract lattices work well when the guards are linear constraints and the assignments are also linear.

Bad states. In each location, the user can define a combination of linear constraints representing the bad states.

Masks. The user can define three kinds of masks:

1. two locations (or more) give the same observation: in this case, the controller is not sure about the exact location of the system.
2. some variables are hidden: the controller cannot determine the values of those variables
3. the value of a numerical variable is unknown if this value belongs to a specified interval. This mask implements variables that are *partially hidden*.

Masks are optional. If there is not any mask specified, then the analysis is performed on a system under full observation.

Non-blocking. SMACS does not ensure that the resulting STS is non-blocking by default; the user must call the program with the option *-ensure_nonblocking* to be effective. We implemented the first method of section 4.2, the one which computes the fixpoint of equation 7:

$$\mathrm{Coreach}_{uc}^{nb}(Bad) = \mathrm{lfp}(\lambda B.Bad \cup \mathrm{Pre}_{bl}(\mathrm{Coreach}_{uc}(B))) \tag{13}$$

Output. The result of SMACS is a description of the controlled system, written in the same syntax as its input.

6.2 Experiments

We experimented our tool on some examples:

- a toy example that illustrates the application of algorithms given in sections 4 and 5; in this example, the mask is defined by a set of locations.
- an example of a cat and a mouse, which was already presented in this paper; in this example, the mask is defined by an hidden variable.
- an example of a shared resource with multiple readers and writers, presented in section6; in this example, there is no mask, *i.e.* the system is under full observation.
- an example with three trains that must not collide; in this example, the mask is defined by intervals.

In all those examples, there exists a most permissive controller and SMACS finds all of them, in less than 20 ms.

Toy example This example is a system with two variables, x and y (see Fig. 5). Location 6 represents the set of bad states. This example features:

- a loop of uncontrollable events, which implies that a naive exact computation of the fixpoint is inefficient;
- two controllable events that must be both disabled because of a masking issue;
- a loop of controllable events that will be disabled when we ensure that the controlled system is non-blocking.

In the easiest case (we observe everything and do not care about blocking), the computation of $I(Bad)$ first detects that x must not be equal to 0 in location 2 because of the uncontrollable event u_0, and this computation terminates by finding that the set $I(Bad)$ is given by:

- $0 \leq x \leq 1000$ in location 2;
- every value of x and y in location 6.

The controller thus disables the transition between locations 3 and 2 when $0 \leq x \leq 1000$.

If the controller must ensure that the system is non-blocking, location 2 becomes "bad" without any condition on x. The controller thus totally disables the event c_0 in location 3.

The partial observation, on this example, is that locations 3 and 4 return the same observations. When the controller must ensure that the system is non-blocking, it also disables the event c_0 in location 4, and thus the whole sequence $0 \xrightarrow{c_2} 1 \xrightarrow{u_1} 4 \xrightarrow{c_0} 5 \xrightarrow{c_2} 3$. SMACS obtained this result in 16 ms.

Cat and Mouse We already presented this example on Fig. 1: a cat and a mouse are in a very big house with more than 1000 rooms, with doors between rooms i and $i + 1$. The cat sometimes sleeps and when she is awake, either she is in the same room than the mouse and eats him, or she moves to the next room. The mouse can only run away. We also put a trap, with cheese, in room 0. If the mouse smells cheese (it is only possible if the mouse is in rooms 0 to 1000), he runs to the room 0 and dies. Controllable transitions are those drawn in plain lines.

The positions of the mouse and of the cat are respectively given by the variables x and y. The bad states correspond to those where the mouse is dead. The controller thus prevents the cat to wake up when she is in the same room than the mouse, and it also disables the event "up2rooms" when $x \leq 1000$.

This example also shows the restrictions made by the controller when we must deal with blocking states and partial observation. When we want to ensure that the controlled system is non-blocking, we disable the event "up2rooms" without any condition. When we ignore the position of the cat (y is an hidden variable), the controller simply prevents the cat to wake up. SMACS obtained this result in 12 ms.

Readers and writers A file is shared between several readers and writers (there is no bound on the number of readers and writers). Several readers can access the file at the same time if no one is writing, and several writers cannot work simultaneously. The set of bad states is expressed by the formula: $(nw = 1 \wedge nr \geq 1) \bigvee (nw \geq 2)$, where nw is the number of writers, and nr the number of readers. The system can add a writer or a reader (actions **startW**,**startR**), or remove a writer, resp. a reader, if $nw \geq 1$ (action **endW**), resp. $nr \geq 1$ (action **endR**). All actions are controllable and the controller observes everything. The controller automatically finds the best solution, preventing to add a reader or a writer when there is already a writer, and preventing to add a writer when there is already a reader. SMACS obtained this result in 16 ms. Note that we obtain the best solution only when we compute the fixpoint within the lattice of convex polyhedra. If we employ the lattice of intervals, the over-approximations are too rough and we obtain a controlled system that cannot add readers nor writers.

Trains Three trains move on three different railroads. They move at the same speed (1 "position" per second) and the first train crosses the path of the two other trains. Before these crossings, there are two stops where one can ask the first train to stay. Other moves are uncontrollable.

This system is modelled by the STS depicted on Fig. 6: the three variables T_1,T_2 and T_3 represent the position of the three trains. The first train can stop at positions 5 and 15. The crossroads are respectively:

- when the first two trains are at position 8,
- when the first train is at position 23 and the third at position 26.

In order to avoid any collision, the controller must stop the first train when the second crosses its road. If the position of the second train is partially unknown, because of a mask defined by the interval $(T_2,[6,10])$, *i.e.* T_2 is not observable when $6 \leq T_2 \leq 10$, then the controller forces the first train to wait 3 seconds longer. SMACS obtained this result in 8 ms.

7 Conclusion and Future Works

We have proposed algorithms for the synthesis of memoryless controllers through partial observation of infinite state systems modelled by STS. One can notice that our algorithm can be used to verify *safety properties*, because a safety problem can be reduced to a state avoidance control problem (see [15] for details). To our knowledge, the improved version of our algorithm provides a better solution than what was previously proposed in the literature with the hypothesis taken. Our tool SMACS implements our algorithms and allowed us to make an empirical validation of our methods and shows its feasibility and usability. For infinite systems, our algorithms use abstract interpretation techniques that provide an over-approximation of the set $I(Bad)$. Further works will look at

possible refinements in the abstract domain to obtain, when needed, more permissive controllers. We will study the synthesis of controllers with memory to provide even more permissive controllers. We also want to study the problem when liveness properties must be fulfilled.

References

1. The APRON library. http://apron.cri.ensmp.fr/.
2. F. Bourdoncle. *Sémantiques des Langages Impératifs d'Ordre Supérieur et Interprétation Abstraite*. PhD thesis, Ecole Polytechnique, 1992.
3. C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
4. K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games of incomplete information. *Logical Methods in Computer Science*, 3(3:4), 2007.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
6. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96, New York, NY, USA, 1978. ACM Press.
7. Fixpoint: an OCaml library implementing a generic fix-point engine. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint/.
8. N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.
9. T.A. Henzinger, R. Majumdar, and J.-F. Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Logic*, 6(1):1–32, 2005.
10. R.C. Hill, D.M. Tilbury, and S. Lafortune. Covering-based supervisory control of partially observed discrete event systems for state avoidance. In *9th International Workshop on Discrete Event Systems*, May 2008.
11. L.E. Holloway, B.H. Krogh, and A. Giua. A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Application*, 7:151–190, 1997.
12. B. Jeannet. Dynamic partitioning in linear relation analysis. Application to the verification of reactive systems. *Formal Methods in System Design*, 23(1):5–37, July 2003.
13. R. Kumar, V. Garg, and S.I. Marcus. Predicates and predicate transformers for supervisory control of discrete event dynamical systems. *IEEE Trans. Autom. Control*, 38(2):232–247, 1993.
14. R. Kumar and V.K. Garg. On computation of state avoidance control for infinite state systems in assignment program model. *IEEE Transactions on Automation Science and Engineering*, 2(2):87–91, 2005.
15. T. Le Gall, B. Jeannet, and H. Marchand. Supervisory control of infinite symbolic systems using abstract interpretation. In *44nd IEEE Conference on Decision and Control (CDC'05) and Control and European Control Conference ECC 2005*, December 2005.

16. A. Miné. The octagon abstract domain. In *Proc. of the Workshop on Analysis, Slicing, and Transformation (AST'01)*, IEEE, pages 310–319, Stuttgart, Gernamy, October 2001. IEEE CS Press.
17. The programming language Objective CAML. http://caml.inria.fr/.
18. P.J. Ramadge and W.M. Wonham. Modular feedback logic for discrete event systems. *SIAM J. Control Optim.*, 25(5):1202–1218, September 1987.
19. P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.
20. S. Takai and S. Kodama. Characterization of all m-controllable subpredicates of a given predicate. *International Journal of Control*, 70:541–549(9), 10 July 1998.
21. Wonham W.M. and P.J. Ramadge. Modular supervisory control of discret-event systems. *Mathematics of Control, Signals, and Systems*, 1(1):13–30, 1988.

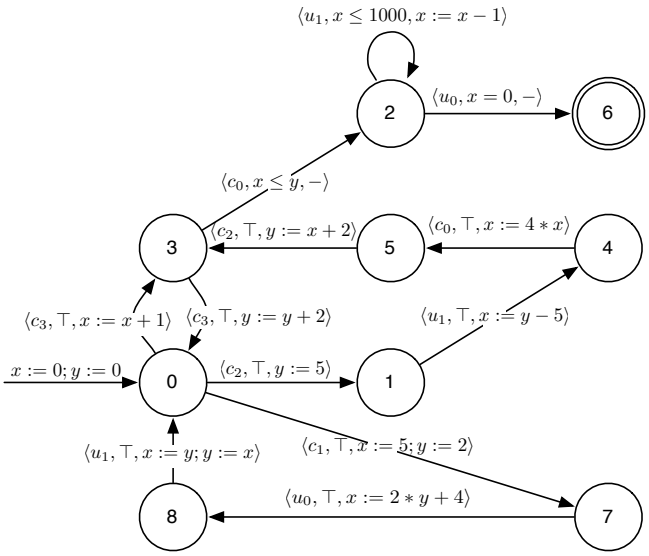

Fig. 5. Toy example

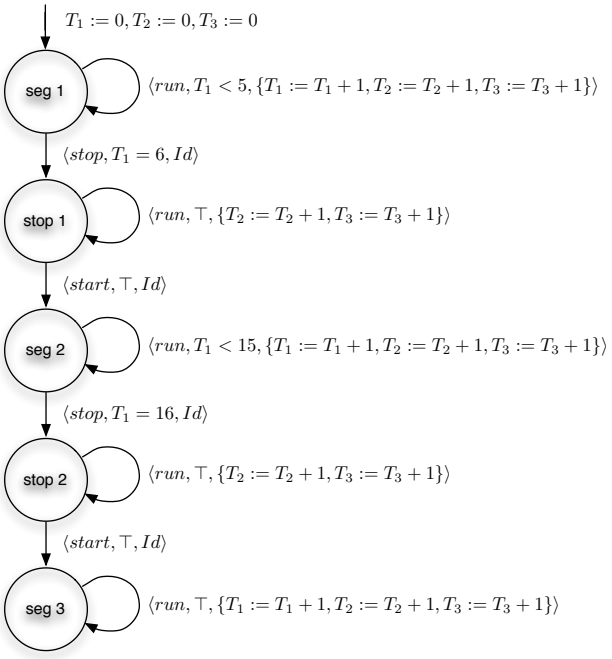

Fig. 6. The Trains Example