Symbolic Supervisory Control of Infinite Transition Systems under Partial Observation using Abstract Interpretation

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

Discrete Event Dynamic Systems

Symbolic Supervisory Control of Infinite Transition Systems under Partial Observation using Abstract Interpretation

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

Abstract We propose algorithms for the synthesis of state-feedback controllers with partial observation of infinite state discrete event systems modelled by Symbolic Transition Systems. We provide models of safe memoryless controllers both for potentially deadlocking and deadlock free controlled systems. The termination of the algorithms solving these problems is ensured using abstract interpretation techniques which provide an overapproximation of the transitions to disable. We then extend our algorithms to controllers with memory and to online controllers. We also propose improvements in the synthesis of controllers in the finite case which, to our knowledge, provide more permissive solutions than what was previously proposed in the literature. Our tool SMACS gives an empirical validation of our methods by showing their feasibility, usability and efficiency.

Keywords Symbolic Transition Systems · Controller Synthesis · Partial Observation · Abstract Interpretation.

1 Introduction

Discrete event systems control theory, as stated by Ramadge and Wonham (1989), provides methods to synthesize controllers which usually have a full observation of the plant modelled by a finite state system, and can disable controllable actions to satisfy control requirements. 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 it has only a *partial observation* of the system, because these devices have a finite precision or because some parts of the plant are not observed by the controller. In this paper, we consider a controller partially observing the system through its states (as opposed to the partial observation of the events). Moreover, an extended model with variables may be better suited to specify the plant. In that case, to provide a homogeneous treatment of these models, it is convenient to consider infinite variables domains.

In this paper, we address the *state avoidance control problem* as defined by Takai and Kodama (1998), where the controller's goal consists in preventing the system from reaching a specified set of states *Bad*; we consider the case of partially observed infinite state systems where the plant is modelled by a Symbolic Transition System (STS) as defined by Henzinger et al (2005). STS is a model of systems defined over a set of variables, whose domain can be infinite, and is composed of a finite set of *symbolic transitions*. Each transition is *guarded* on the system variables, and has an *update* function which indicates the variables

G. Kalyon is 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.

G. Kalyon · T. Le Gall · T. Massart
Université Libre de Bruxelles (U.L.B.), Campus de la Plaine, Bruxelles, Belgique
E-mail: {gkalyon,tlegall,tmassart}@ulb.ac.be

H. Marchand
INRIA, Centre Rennes - Bretagne Atlantique
E-mail: herve.marchand@inria.fr

changes which occur 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[1]. Since 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 as done by Wonham and Ramadge (1988). Moreover, to take into account the partial observation of the controller, we follow and extend the approach taken by Kumar et al (1993), where the partial observation is modelled by a *mask* corresponding to a mapping from the state space to a (possibly) *infinite* observation space. Note that an important topic is the quality of the synthesized controller: it can be measured by *permissiveness* criteria as discussed by Takai and Kodama (1998), which state, for example, that the set of transitions allowed by the controller must be maximal.

Related works. The *Controller synthesis of symbolic finite state systems* has been considered e.g. by Balemi et al (1993); Marchand et al (2000); Miremadi et al (2008a). The idea, as developed by Kumar et al (1993), was to encode subsets of the state-space by predicates and all the operations on state sets by predicate transformers, thus avoiding the enumeration of the state space while computing the supervisors[2].

The *controller synthesis of finite state systems with partial observation* on the actions has also been widely studied in the literature (see e.g. Lin and Wonham (1988); Brandt et al (1990); Takai and Ushio (2003); Yoo and Lafortune (2006) and Cassandras and Lafortune (2008) for a review). The problem with partial observation on the states has been introduced by Kumar et al (1993) using the notion of mask, which is defined as a partition of the state space (i.e. disjoint sets of indistinguishable states). Takai and Kodama (1998) define the notion of *M-controllability* and they present necessary and sufficient conditions for the existence of a controller whose resulting controlled system allows to reach *exactly* a set of allowable states Q. Hill et al (2008) extend this work and provide a method which synthesizes more permissive controllers, where the mask used is a covering of the state space (i.e. overlapping sets of indistinguishable states): the observation mask, through which the controller observes the system state, provides at any time a single observation among the possible observations of the current state. Kumar et al (1993) propose an online algorithm, which consists in computing at each step of the system's execution, the actions to be forbidden according to the possible current states. This algorithm is only defined for finite systems. *Controller synthesis of infinite state systems in the case of full observation* has also been studied in several works. Kumar and Garg (2005) extend their previous work (Kumar et al (1993)) to consider infinite systems and they prove that the state avoidance control problem is then undecidable. They also show that the problem can be solved in the case of Petri nets, when the set *Bad* is upward-closed. Le Gall et al (2005) use, symbolic techniques to control infinite systems modelled by STS[3]. Abstract interpretation techniques as defined by Cousot and Cousot (1977) are then used by Le Gall et al (2005) to ensure that the controlled system can be effectively computed. These methods have been extended by Kalyon et al (2009) to take into account partial observation; the masks used are partitions of the state space. Le Gall et al (2005); Kalyon et al (2009) consider *memoryless* controller where the control decision is only based on the current state of the system.

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 (as done by Pnueli and Rosner (1989); Reif (1984a); De Wulf et al (2006a); Chatterjee et al (2007); Kupferman et al (2000)). Reif (1984b) studies games of *incomplete information* (i.e. the set of observations is a partition of the state space) for finite systems. He proposes an algorithm that first transforms the game of incomplete information into a game of *perfect information* using a determinization procedure and that then solves this game. De Wulf et al (2006b) study games of *imperfect information* (i.e. the set of observations is a covering of the state space) for finite systems with safety objectives. They propose a fixpoint theory, which is defined on the lattice of antichains of sets of states, to solve efficiently the problem without determinization procedure. Chatterjee et al (2007) extend the work of De Wulf et al (2006b) considering ω-regular objectives (see also Thistle and Lamouchi (2009) for the control of ω-regular properties under event-based partial observation).

Contributions of this paper. In this paper, we extend Kalyon et al (2009) in two ways: we consider masks that are coverings of the state space instead of partitions, and we improve the permissiveness of the control algorithm. In order to deal with the infiniteness of state space, the algorithms presented in this

[1] Miremadi et al (2008b) present a similar model but the domain of the variables is assumed to be finite.

[2] The underlying structure used to encode the sets of states is mostly based on a data structure named Binary Decision diagram defined by Bryant (1986), that has been proved to be very efficient to encode boolean predicates

[3] Note that Jeannet et al (2005) assume this alphabet to be infinite.

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 as done by Balemi et al (1993); Marchand et al (2000); Kumar et al (1993). Moreover, since the problem is undecidable as shown by Le Gall et al (2005), we keep using abstract interpretation techniques to get effective algorithms (i.e. which always terminate). It is worth noticing that both *concrete and abstract domains can be infinite*. It is important to note that using these abstract interpretation techniques leads us to redefine the algorithms defined, e.g. by Takai and Kodama (1998), in the finite case so that only over-approximations are involved while computing the controller. Next, we provide methods to improve the quality of the synthesized controllers and we proposed two different solutions: the k-memory controller and the online controllers. These controllers keep a part or the totality of the execution of the system and use this information to define their control policy. Moreover, we theoretically compared our different controllers; it results that the online controllers define the best control policy and that the k-memory controllers are better than the memoryless controllers. Finally, algorithms for the (non-blocking) memoryless control problem have been implemented in the tool SMACS (SMACS (2010)) to validate them experimentally on some benchmarks.

In section 2, we introduce our model for infinite systems to be controlled. In section 3, we define the control mechanisms used and the state avoidance control problem. In section 4, we present an algorithm, which solves our problem, but does not always terminate and explain how to obtain an effective algorithm using abstract interpretation techniques. In section 5, we define a controller which ensures that the controlled system satisfies the deadlock free property. In section 6, we give two methods to improve the quality of the controller (controllers with memory and online controllers). In section 7, we briefly present SMACS and provide examples that experimentally validate our method.

2 Symbolic Transition Systems

When modelling realistic systems, it is often convenient to manipulate state variables instead of simply atomic states. Within this framework, the states can be seen as a particular instantiation of vector of variables. In this case, the techniques and associated tools derived from the labelled transition systems do not explicitly take into account the data as the underlying model of transition systems which implies that the variables must be instantiated during the state space exploration and analysis. This enumeration of the possible values of the variables leads to the classical state space explosion when these variables take their value in a finite set, but may also render the computation infeasible whenever the domain of the variables is infinite. The model of Symbolic Transition Systems (STS) is a transition system with variables, whose domain can be infinite, and is composed of a finite set of *symbolic transitions*. Each transition is *guarded* on the system variables, and has an *update* function which indicates the variables changes when the transition is fired. Transitions are labelled with symbols taken from a finite alphabet. This model allows to represent infinite systems whenever the variables take their values in an infinite domain, while it has a finite structure and offers a compact way to specify systems handling data.

Variables, Predicates, Assignments. In the sequel, we shall assume a tuple of typed variables. The (infinite) domain of a variable v is denoted $\mathcal{D}_v$. If $V = \langle v_1, \ldots, v_n \rangle$ is a n-tuple (n is constant) of variables, we note $\mathcal{D}_V = \prod_{i \in [1,n]} \mathcal{D}_{v_i}$. A *valuation* $\boldsymbol{\nu}$ of V is a n-tuple $\langle \boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_n \rangle \in \mathcal{D}_V$. A *predicate* over a n-tuple V is defined as a subset $P \subseteq \mathcal{D}_V$ (a set of states for which the predicate holds). The complement of a set $H \subseteq \mathcal{D}_V$ is denoted by $\overline{H}$. The preimage function of $f : D_1 \mapsto D_2$ is denoted by $f^{-1} : D_2 \mapsto 2^{D_1}$ and is defined, for all $d_2 \in D_2$, by $f^{-1}(d_2) = \{d_1 \in D_1 | f(d_1) = d_2\}$. We naturally extend a function $f : D_1 \mapsto D_2$ to sets $H \subseteq D_1$ as follows: $f(H) = \bigcup_{h \in H} f(h)$. Given a n-tuple $\langle \boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_i, \ldots, \boldsymbol{\nu}_n \rangle \in \mathcal{D}_{v_1} \times \ldots \times \mathcal{D}_{v_i} \times \ldots \times \mathcal{D}_{v_n}$, we define the function Proj_i as the projection on the i^{th} element of this tuple, i.e. $\mathsf{Proj}_i(\langle \boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_i, \ldots, \boldsymbol{\nu}_n \rangle) = \boldsymbol{\nu}_i$. When no confusion is possible (i.e. $\forall j \in [1,n] : j \neq i \Rightarrow \mathcal{D}_{v_i} \neq \mathcal{D}_{v_j}$), we denote Proj_i by $\mathsf{Proj}_{\mathcal{D}_{v_i}}$. Note that $\mathcal{D}_{v_i}$ can itself be a tuple $\prod_{j \leq k} \mathcal{D}'_{v'_j}$.

Definitions. Let us now formally define the Symbolic Transition Systems.

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 n-tuple of variables (n is constant)*

– $\Theta \subseteq \mathcal{D}_V$ *is a predicate on* V*, which defines 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*, which guards* δ*,*
  – $A_\delta : \mathcal{D}_V \mapsto \mathcal{D}_V$ *is the update function of* δ*.* •

Throughout this thesis, we suppose that the update functions A_δ of an STS are *continuous* to ensure the existence of a least fixpoint in the fixpoint equations that we will define.

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* $[\![\mathcal{T}]\!] = \langle X, X_0, \Sigma, \rightarrow \rangle$ *where:*

– $X = \mathcal{D}_V$ *is the set of states*
– $X_0 = \Theta$ *is the set of initial states*
– Σ *is the set of labels*
– $\rightarrow \subseteq X \times \Sigma \times X$ *is the transition relation defined as* $\{\langle \boldsymbol{\nu}, \sigma, \boldsymbol{\nu}' \rangle \mid \exists \delta \in \Delta : (\sigma_\delta = \sigma) \wedge (\boldsymbol{\nu} \in G_\delta) \wedge (\boldsymbol{\nu}' = A_\delta(\boldsymbol{\nu}))\}$. •

In any state, a transition can be fired only if its guard is satisfied; in this case, the variables are updated according to the update function. Note that the LTS $[\![\mathcal{T}]\!]$ can be non-deterministic. A state $\boldsymbol{\nu} \in \mathcal{D}_V$ is in *deadlock* if no transition can be fired from $\boldsymbol{\nu}$, i.e. $\forall \delta \in \Delta : \boldsymbol{\nu} \notin G_\delta$. An *execution* or a *run* of an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$ is given by a sequence $\boldsymbol{\nu}_0 \xrightarrow{\sigma_0} \boldsymbol{\nu}_1 \ldots \xrightarrow{\sigma_n} \boldsymbol{\nu}_{n+1}$ accepted by $[\![\mathcal{T}]\!]$ and a *trace* of this run is given by $\sigma_0 \ldots \sigma_n$.

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

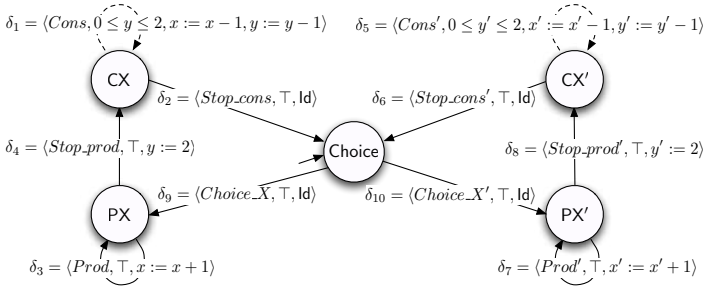

Fig. 1: Producer and consumer

Example 1 *The STS of Fig. 1 is a system of stock management.* Id*,* $\top$ *and* $\bot$ *denote respectively the identity function, and the predicates* true *and* false*. Two units produce and send (consume) two kinds of pieces* X *and* X'*. The STS has explicit locations* $\ell \in \{\mathsf{CX}, \mathsf{PX}, \mathsf{Choice}, \mathsf{CX}', \mathsf{PX}'\}$ *and four natural variables:* x *(resp.* x'*) gives the number of pieces* X *(resp.* X'*) and* y *(resp.* y'*) gives the number of pieces* X *(resp.* X'*) that can be consumed. A state system corresponds to a* 5*-tuple* $\langle \ell, x, x', y, y' \rangle$*. The initial state is* $\langle \mathsf{Choice}, 50, 50, 0, 0 \rangle$*. In the location* CX *(resp.* CX′*), the action Cons (resp. Cons*′*) allows to consume a piece* X *(resp.* X'*) and Stop_cons (resp. Stop_cons*′*) allows to stop the process of consumption. In the location* PX *(resp.* PX′*), the action Prod (resp. Prod*′*) allows to produce a piece of kind* X *(resp.* X'*) and Stop_prod (resp. Stop_prod*′*) allows to stop the process of production. The choice of the kind of pieces to produce is done in the location* Choice*. The variables* y *and* y' *ensure that at most three pieces can be consumed in each cycle of consumption.* ◇

Notations and Predicate Transformers. In the sequel, we shall use the following notations, for an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$ and for all $\delta = \langle \sigma_\delta, A_\delta, G_\delta \rangle \in \Delta$, $D \subseteq \Delta$, $\sigma \in \Sigma$, $A \subseteq \Sigma$ and $B \subseteq \mathcal{D}_V$:

- $\text{Trans}(\sigma) = \{\delta \in \Delta \mid \sigma_\delta = \sigma\}$ is the set of transitions labelled by σ.
- $\mathcal{L}(\mathcal{T}) \subseteq \Sigma^*$ is the set of traces associated to $\mathcal{T}$.
- $\text{Pre}_\delta(B) = G_\delta \cap A_\delta^{-1}(B) = \{\boldsymbol{\nu} \in \mathcal{D}_V \mid \exists \boldsymbol{\nu}' \in B : (\boldsymbol{\nu} \in G_\delta) \wedge (\boldsymbol{\nu}' = A_\delta(\boldsymbol{\nu}))\}$ is the set of states leading to B through the transition δ.
- $\text{Post}_\delta(B) = A_\delta(G_\delta(B))$ is the set of states that are reachable from B through the transition δ.
- $\text{Pre}_D(B) = \bigcup_{\delta \in D}(\text{Pre}_\delta(B))$ is the set of states leading to B through a transition in D.
- $\text{Post}_D(B) = \bigcup_{\delta \in D}(\text{Post}_\delta(B))$ is the set of states that are reachable from B through a transition in D.
- $\text{Pre}_A(B) = \bigcup_{\sigma \in A}(\text{Pre}_{\text{Trans}(\sigma)}(B))$ is the set of states leading to B through a transition labelled by an action in A.
- $\text{Post}_A(B) = \bigcup_{\sigma \in A}(\text{Post}_{\text{Trans}(\sigma)}(B))$ is the set of states that are reachable from B through a transition labelled by an action in A.
- $\mathsf{reachable}(\mathcal{T}) \subseteq \mathcal{D}_V$ is the set of states that are reachable from the initial states of $[\![\mathcal{T}]\!]$.

Note that throughout this paper, we work with sets of states and use operations on these sets. In our tool, the set of states are symbolically represented by predicates. Each operation on sets is performed by a predicate transformer (for example $\text{Pre}_\delta(B)$ is given by the set of states $\boldsymbol{\nu}$, which satisfy the predicate transformer $\exists \boldsymbol{\nu}' \in B : (\boldsymbol{\nu} \in G_\delta) \wedge (\boldsymbol{\nu}' = A_\delta(\boldsymbol{\nu}))$. Details can be found in Le Gall et al (2005) and in Jeannet et al (2005).

Safety properties and observers. Given an STS, an *invariance property* is defined by a set of states E such that from any state in that set, it is not possible with any allowed transition of the system to leave E. Dually, one can define a set of states *Bad* that must not be reached along the execution of the system. We have opted for this approach here and refer to it as a *forbidden state invariance property Bad*. Now, one can also want to specify more general properties related to the notion of traces of the system. In this setting, a *safety property* P is a set of traces such that $\rho \notin P \Rightarrow \forall \rho' : \rho \cdot \rho' \notin P$. In other words, as soon as the property is not satisfied, it is false forever. From a computational point of view, it is always possible to reduce a safety property to a forbidden state invariance property by means of *observers* (this notion of observer is different from the one given in Def. 3 to model the partial observation). Given a (deterministic) STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, an observer is a structurally deterministic[4] STS $\mathcal{O}$ which is *non-intrusive*, i.e. an observer verifies $\mathcal{L}(\mathcal{T} \times \mathcal{O}) = \mathcal{L}(\mathcal{T})$[5]. The observer $\mathcal{O}$ is equipped with a dedicated variable $\mathsf{Violate}$ such that once this variable becomes true, then it is true forever and the *language* $\mathcal{L}_{\mathsf{Violate}}(\mathcal{O})$ *recognized by* $\mathcal{O}$ is the set of traces of runs that contain states in which the boolean variable $\mathsf{Violate}$ is true (the set of states in which $\mathsf{Violate}$ is true, will be denoted by $\mathsf{Violate}^{\mathcal{O}}$). In fact, an observer encodes the negation of a safety property $\varphi_{\mathcal{O}}$ and $\mathcal{L}_{\mathsf{Violate}}(\mathcal{O})$ corresponds to the set of traces that violate $\varphi_{\mathcal{O}}$. We have (Le Gall et al (2005)):

Proposition 1 *Let $\mathcal{T}$ be an STS and $\mathcal{O}$ an observer for $\mathcal{T}$ defining the safety property $\varphi_{\mathcal{O}}$. Then $\mathcal{T}$ satisfies $\varphi_{\mathcal{O}}$ if and only if $\mathcal{T} \times \mathcal{O}$ satisfies the forbidden state invariance property $\mathcal{D}_V \times \mathsf{Violate}^{\mathcal{O}}$.*

Based on Prop. 1, we will only consider forbidden state invariance property in the remainder of the paper. Indeed, controlling $\mathcal{T}$ to ensure the safety property $\varphi_{\mathcal{O}}$ modelled by $\mathcal{O}$ is equivalent to controlling $\mathcal{T} \times \mathcal{O}$ to ensure the forbidden state invariance property $\mathcal{D}_V \times \mathsf{Violate}^{\mathcal{O}}$.

3 The State Avoidance Control Problem

We can now define the state avoidance control problem w.r.t. the available information from the observation of the system and the available control mechanisms.

3.1 Means of Observation

The controller interacts with the plant through *sensors* and *actuators* and it has generally only a *partial observation* of the system, because these devices have not an absolute precision or some parts of the system are not observed by the controllers. To model the partial observation that the controller has of the current state of the system, we use the concept of *observer* defined as follows:

[4] An STS is *structurally deterministic* if $\forall \delta_1, \delta_2 \in \Delta : (\sigma_{\delta_1} = \sigma_{\delta_2}) \Rightarrow (G_{\delta_1} \cap G_{\delta_2} = \varnothing)$. The structural determinism of an STS $\mathcal{T}$ implies that the corresponding LTS $[\![\mathcal{T}]\!]$ is deterministic.

[5] Where $\times$ denotes the classical synchronous product between STS such that $\mathcal{L}(\mathcal{T}_1 \times \mathcal{T}_2) = \mathcal{L}(\mathcal{T}_1) \cap \mathcal{L}(\mathcal{T}_2)$ (see Jeannet et al (2005) for details).

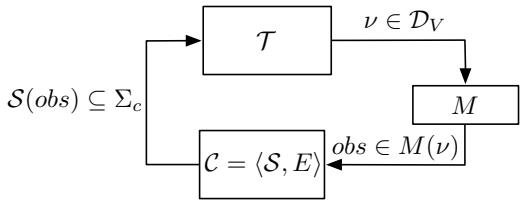

Fig. 2: Feedback interaction between the system and the controller

Definition 3 (Observer) *An observer of the state space $\mathcal{D}_V$ is a pair $\langle Obs, M \rangle$, where Obs is a variable, whose domain is the (possibly infinite) observation space $\mathcal{D}_{Obs}$ and the mask $M : \mathcal{D}_V \cup \{\varnothing\} \mapsto 2^{\mathcal{D}_{Obs}}$ gives, for each state $\boldsymbol{\nu} \in \mathcal{D}_V$, the (possibly infinite) set $M(\boldsymbol{\nu})$ of possible observations one can have when the system enters this state. Moreover, it is imposed that each state $\boldsymbol{\nu} \in \mathcal{D}_V$ has at least one observation (i.e. $\forall \boldsymbol{\nu} \in \mathcal{D}_V : M(\boldsymbol{\nu}) \neq \varnothing$) and we assume that $M(\varnothing) = \varnothing$.* •

Note that we have chosen to define the observation space $\mathcal{D}_{Obs}$ as the domain of the variable Obs to remain consistent with the formalization of the state space $\mathcal{D}_V$.

We say that a state $\boldsymbol{\nu} \in \mathcal{D}_V$ is *compatible* with the observation $obs \in \mathcal{D}_{Obs}$, if $obs \in M(\boldsymbol{\nu})$. For each observation $obs \in \mathcal{D}_{Obs}$, $M^{-1}(obs)$ gives the set of states compatible with this observation. One can notice that the mask M is a *covering* of the state space. In this setting, a *full observation* is a particular case of partial observation defined by $Obs = V$ and M is equal to the identity function. Moreover, a mask M defined by a *partition* of the state space is a particular case of the covering one where each state $\boldsymbol{\nu} \in \mathcal{D}_V$ is compatible with exactly one observation state.

Example 2 *For the system of Fig. 1, an example of partial observation is given by the mask $M : Loc \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mapsto 2^{Loc \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}}$, where for each state $\boldsymbol{\nu} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$ the value of all possible observations of $\boldsymbol{\nu}$ the controller can have is given by*

$$M(\langle \ell, x, x', y, y' \rangle) = \begin{cases} \{\langle \ell, x_1, x', y, y' \rangle | (x_1 \in [x-1, x+1])\} & \text{if } x \geq 1 \\ \{\langle \ell, x_1, x', y, y' \rangle | (x_1 \in [x, x+1])\} & \text{if } x = 0 \end{cases}$$

modelling the fact that there is an imprecision on the number of pieces X. ◇

3.2 Means of Control

Following the theory developped by Ramadge and Wonham (1989) (see also Cassandras and Lafortune (2008)), we add a controller $\mathcal{C}$ to the system $\mathcal{T}$. These elements interact in a feedback manner, as illustrated in Fig. 2: the controller observes the system and according to its observation delivers the set of events that have to be disabled in order to ensure the desired properties on the system. The control is performed by means of *controllable events*: the alphabet Σ is partitioned into the set of controllable events Σ_c and the set of uncontrollable events Σ_{uc}; only controllable events can be forbidden by the controller. As a consequence, the set of symbolic transitions Δ is also partitioned accordingly into Δ_c and Δ_{uc} where $\langle \sigma, G, A \rangle \in \Delta_c$ (resp. Δ_{uc}) if $\sigma \in \Sigma_c$ (resp. Σ_{uc}).

3.3 Controller and Controlled System

In this paper, the controller aims at restricting the system's behavior to ensure a forbidden state invariance property (i.e. to prevent the system from reaching forbidden states). The controller with partial information is formally defined as follows:

Definition 4 (Controller) *Given an STS $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$, and an observer $\langle Obs, M \rangle$, a controller for $\mathcal{T}$ is a pair $\mathcal{C} = \langle \mathcal{S}, E \rangle$, where:*

1. *$\mathcal{S} : \mathcal{D}_{Obs} \mapsto 2^{\Sigma_c}$ is a supervisory function which defines, for each observation $obs \in \mathcal{D}_{Obs}$, a set $\mathcal{S}(obs)$ of controllable actions to be forbidden when obs is observed by the controller.*
2. *$E \subseteq \mathcal{D}_V$ is a set of states to be forbidden, which restricts the set of initial states.* •

At each step of the system's execution, the controller gets a single observation among the set of possible observations of the current state. This observation is maintained until the arrival of next one (i.e. the controller is *memoryless*).

In the sequel, to avoid repetitions, we suppose to work with a system $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$ to be controlled, an observer $\langle Obs, M \rangle$, a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ and a predicate Bad, which gives the forbidden states.

In this framework, the controlled system is not explicitly computed, but its semantics is characterized by an STS where the variable Obs provides the observations the controller gets from the system.

Definition 5 (Controlled STS) *The system $\mathcal{T}$ controlled by $\mathcal{C}$, is an STS including observations $\mathcal{T}_{/\mathcal{C}} = \langle V_{/\mathcal{C}}, \Theta_{/\mathcal{C}}, \Sigma, \Delta_{/\mathcal{C}} \rangle$, where:*

- *$V_{/\mathcal{C}}$ is a tuple of variables, whose domain is $\mathcal{D}_V \times \mathcal{D}_{Obs}$.*
- *$\Theta_{/\mathcal{C}} = \{\langle \boldsymbol{\nu}, obs \rangle | (\boldsymbol{\nu} \in (\Theta \setminus E)) \wedge (obs \in M(\boldsymbol{\nu}))\}$ is the initial condition.*
- *$\Delta_{/\mathcal{C}}$ is defined using the following rule:*

$$\langle \sigma, G, A \rangle \in \Delta \quad \mathcal{O}_\sigma = \{obs \in \mathcal{D}_{Obs} | \sigma \notin \mathcal{S}(obs)\}$$

$$G_{/\mathcal{C}} = \{\langle \boldsymbol{\nu}, obs \rangle | (\boldsymbol{\nu} \in G) \wedge (obs \in (\mathcal{O}_\sigma \cap M(\boldsymbol{\nu})))\}$$

$$\forall \boldsymbol{\nu} \in \mathcal{D}_V, \forall obs \in M(\boldsymbol{\nu}) : A_{/\mathcal{C}}(\langle \boldsymbol{\nu}, obs \rangle) = \{\langle \boldsymbol{\nu}', obs' \rangle | (\boldsymbol{\nu}' = A(\boldsymbol{\nu})) \wedge (obs' \in M(\boldsymbol{\nu}'))\}$$

$$\langle \sigma, G_{/\mathcal{C}}, A_{/\mathcal{C}} \rangle \in \Delta_{/\mathcal{C}}$$

•

A state $\langle \boldsymbol{\nu}, obs \rangle$ (with $obs \in M(\boldsymbol{\nu})$) of the controlled system models the fact that the controller got the observation obs, when the system was in the state $\boldsymbol{\nu}$. The guards and the update functions of the system are modified to take into account the observations:

1. *Guards:* A transition δ (labelled by σ) can be fired from a state $\langle \boldsymbol{\nu}, obs \rangle$ in the controlled system $\mathcal{T}_{/\mathcal{C}}$, if δ can be fired from $\boldsymbol{\nu}$ in $\mathcal{T}$, $\boldsymbol{\nu}$ is compatible with obs, and σ is not forbidden by control in obs ($\mathcal{O}_\sigma$ is the set of observation states for which the function $\mathcal{S}$ allows to fire σ).
2. *Update functions:* The states $\langle \boldsymbol{\nu}', obs' \rangle$ are reachable from the state $\langle \boldsymbol{\nu}, obs \rangle$ through the transition δ in the controlled system $\mathcal{T}_{/\mathcal{C}}$, if $\boldsymbol{\nu}'$ is reachable from $\boldsymbol{\nu}$ through δ in $\mathcal{T}$, and $\boldsymbol{\nu}'$ is compatible with obs'. Thus, this function is such that if $\langle \boldsymbol{\nu}', obs' \rangle \in A_{/\mathcal{C}}(\langle \boldsymbol{\nu}, obs \rangle)$, then $\langle \boldsymbol{\nu}', obs'' \rangle \in A_{/\mathcal{C}}(\langle \boldsymbol{\nu}, obs \rangle)$ for each $obs'' \in M(\boldsymbol{\nu}')$. This allows to model the fact that when the system arrives in the state $\boldsymbol{\nu}'$, then the controller can receive any observation among those ones in $M(\boldsymbol{\nu}')$.

3.4 Definition of the Control Problems

In the sequel, we shall be interested in distinct versions of the state avoidance control problem that consists in preventing the system from reaching some particular states, either because some properties are not satisfied in these states (e.g. the states where two states variables are equal or more generally states in which some particular state predicates are not satisfied) or because they are deadlocking states.

Problem 1 (Basic state avoidance control problem) *The basic state avoidance control problem (basic problem for short) consists in building a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that $\mathsf{Proj}_{\mathcal{D}_V}(\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}})) \cap Bad = \varnothing$.*

For each element $\langle \boldsymbol{\nu}, obs \rangle \in \mathsf{reachable}(\mathcal{T}_{/\mathcal{C}})$, the projection $\mathsf{Proj}_{\mathcal{D}_V}$ gives the part $\boldsymbol{\nu}$ of this element corresponding to the domain $\mathcal{D}_V$. Thus, the basic problem consists in synthesizing a controller which prevents from reaching any element of Bad. In the sequel, *valid* controller $\mathcal{C}$ denotes a controller such that $\mathsf{Proj}_{\mathcal{D}_V}(\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}})) \cap Bad = \varnothing$.

A solution to the basic problem does not ensure that the controlled system is deadlock free, i.e. it is not ensured that the controlled system has always the possibility to make a move. Since most of the time a deadlocking system cannot be tolerated, we extend our problem to the case where the deadlock free property must be ensured.

Problem 2 (Deadlock free state avoidance control problem) *The deadlock free state avoidance control problem (deadlock free problem for short) consists in defining a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that:*

1. $\mathsf{Proj}_{\mathcal{D}_V}(\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}})) \cap Bad = \varnothing$
2. $\forall \langle \boldsymbol{\nu}, obs \rangle \in \mathsf{reachable}(\mathcal{T}_{/\mathcal{C}}), \exists \delta \in \Delta_{/\mathcal{C}} : \langle \boldsymbol{\nu}, obs \rangle \in (G_\delta)_{/\mathcal{C}}$

We can immediately notice that a class of trivially valid controllers $\langle \mathcal{S}, E \rangle$ are the ones where $E = \Theta$ (i.e. where no state remains). Therefore, the notion of permissiveness has been introduced to compare the quality of different controllers for a given STS.

Definition 6 (Permissiveness) *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 $\mathsf{Proj}_{\mathcal{D}_V}(\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_1})) \supseteq \mathsf{Proj}_{\mathcal{D}_V}(\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_2}))$. When the inclusion is strict, we say that $\mathcal{C}_1$ is strictly more permissive than $\mathcal{C}_2$.* •

Indeed, in our settings, since the observations and the control of the system are based on (masked) states it seems more consistent to define the permissiveness w.r.t the states that are reachable in the controlled system, rather than w.r.t. the language of the actions that can be fired. Notice also that two controlled systems with the same reachable state space can have different enabled transitions[6]. However, it can be shown:

Proposition 2 *In general, there is no most permissive controller solving the basic or deadlock free problems.*

Proof The example of Fig. 3 is sufficient to prove the proposition.

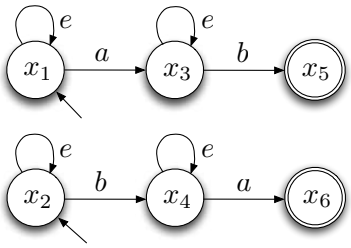

Fig. 3

The set of initial states of the STS $\mathcal{T}_1$ is $X_0 = \{x_1, x_2\}$ and all transitions are controllable[7]. The set $Bad = \{x_5, x_6\}$ and the observer $\langle Obs, M \rangle$ is defined by $\mathcal{D}_{Obs} = \{obs_1, obs_2, obs_3\}$ and:

$$M(x) = \begin{cases} obs_1 & \text{if } x \in \{x_1, x_4\} \\ obs_2 & \text{if } x \in \{x_2, x_3\} \\ obs_3 & \text{if } x \in \{x_5, x_6\} \end{cases}$$

There are three possibilities to avoid the set Bad:

- forbid a in obs_1: $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_1}) = \{\langle x_1, obs_1 \rangle, \langle x_2, obs_2 \rangle, \langle x_4, obs_1 \rangle\}$.
- forbid b in obs_2: $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_2}) = \{\langle x_1, obs_1 \rangle, \langle x_2, obs_2 \rangle, \langle x_3, obs_2 \rangle\}$.
- forbid a in obs_1 and b in obs_2: $\mathsf{reachable}(\mathcal{T}_{/\mathcal{C}_3}) = \{\langle x_1, obs_1 \rangle, \langle x_2, obs_2 \rangle\}$.

No controller is more permissive than all the others, since $\mathcal{C}_1$ and $\mathcal{C}_2$ are both more permissive than $\mathcal{C}_3$, but are not comparable. □

Note that for other kinds of permissiveness like language or execution inclusions, the result would have been the same.

In consequence, we define a maximal solution to the basic (resp. deadlock free) problem as a controller $\mathcal{C}$ solving this problem and such that there does not exist a strictly more permissive controller $\mathcal{C}'$, which also solves this problem. Based on this definition, we define the maximal basic (resp. maximal deadlock free) problem, as the problem which consists in restricting the basic (resp. deadlock free) problem in finding a maximal controller $\mathcal{C}$. Unfortunately, we have the following properties:

Proposition 3 *The maximal basic problem is undecidable.*

Proof Under full observation, computing the maximal controller solving the basic problem is undecidable as shown by Kumar and Garg (2005). Since full observation is a particular case of partial observation, the result follows. □

Proposition 4 *The maximal deadlock free problem is undecidable.*

[6] We could have used an extended definition of permissiveness where if two controlled systems have equal reachable state space, inclusion of the transitions that can be fired from reachable states is also taken into account.

[7] In the sequel, for more clarity and conciseness in some examples, we sometimes directly define the system to be controlled by the LTS which corresponds to the STS modelling it.

Proof The undecidability of the maximal deadlock free problem is proven by a restriction from the maximal basic problem to this one. This restriction consists in building from $\mathcal{T} = \langle V, \Theta, \Sigma, \Delta \rangle$ the system $\mathcal{T}' = \langle V, \Theta, \Sigma', \Delta' \rangle$, where:

- $\Sigma' = \Sigma \cup \{\tau\}$. The new action τ does not belong to Σ and is uncontrollable.
- $\Delta' = \Delta \cup \{\delta'\}$, where the uncontrollable transition $\delta' = \langle \tau, \mathcal{D}_V, \mathsf{Id} \rangle$. This transition can be fired from any state $\boldsymbol{\nu} \in \mathcal{D}_V$ and loops on $\boldsymbol{\nu}$.

$\mathcal{T}'$ does not contain deadlock states. Thus, a controller is a solution to the maximal deadlock free problem if and only if it is a solution to the maximal basic problem. □

By Prop. 3 (resp. 4), it is clear that no algorithm can compute a maximal controller for the basic (resp. deadlock free) problem and ensure the termination of the computations. Hence, our approach will consist in using approximations to ensure the termination of the computations and our aim is to find in this way solutions that are correct and as close as possible to a maximal solution to be of good practical value. Our experiments will validate our solutions computed with this approach.

4 Computation of a Controller for the Basic Problem

In this section, we present our method, based on abstract interpretation, which synthesizes memoryless controllers for the basic problem. First, in section 4.1, we present a *semi-algorithm* (i.e., which does not always terminate) which computes a memoryless controller for the basic problem. Next, in section 4.2, we explain how to extend it to obtain at the price of overapproximations an *effective algorithm* (i.e., which always terminates).

4.1 Semi-algorithm for the Basic Control Problem

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. Then, based on this set of states, we compute the controller, whose aim is to disable, for each observation $obs \in \mathcal{D}_{Obs}$, 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. Let us formalize the two steps of the construction:

Computation of $I(Bad)$. This set of states and more generally $I(.)$ is given by the function $\mathrm{Coreach}_{uc} : 2^{\mathcal{D}_V} \mapsto 2^{\mathcal{D}_V}$ defined below. It corresponds to the set of states that lead to Bad firing only uncontrollable transitions. Classically, we first define the function $\mathrm{Pre}_{uc}(B)$, which computes the set of states from which a state of B is reachable by triggering exactly one uncontrollable transition.

$$\mathrm{Pre}_{uc}(B) = \mathrm{Pre}_{\Delta_{uc}}(B)$$

$\mathrm{Coreach}_{uc}(Bad)$ is then obtained by the following fixpoint equation:

$$\mathrm{Coreach}_{uc}(Bad) = \mathrm{lfp}(\lambda B.\, Bad \cup \mathrm{Pre}_{uc}(B)) \tag{1}$$

By the Tarski's theorem (Tarski (1955)), since the function $\mathrm{Coreach}_{uc}$ is monotonic, the limit of the fixpoint $\mathrm{Coreach}_{uc}(Bad)$ actually exists. But it may be uncomputable, because the coreachability is undecidable in the model of CFSM. In subsection 4.2, we explain how to compute an overapproximation of this fixpoint by ensuring the termination of the computations.

Computation of the controller $\mathcal{C}$. We first define a function $\mathcal{F} : \Sigma \times 2^{\mathcal{D}_V} \mapsto 2^{\mathcal{D}_{Obs}}$, where for an action $\sigma \in \Sigma$ and a set $B \subseteq \mathcal{D}_V$ of states to be forbidden, $\mathcal{F}(\sigma, B)$ specifies the set of observation states, for which the action σ has to be forbidden, i.e. the smallest set $\mathcal{O}$ of observations such that there exists a state $\boldsymbol{\nu} \in \mathcal{D}_V$ with $\mathcal{O} \cap M(\boldsymbol{\nu}) \neq \varnothing$, from which a transition labelled by σ leads to B (see Fig. 4).

$$\mathcal{F}(\sigma, B) = \begin{cases} M(\mathrm{Pre}_\sigma(B) \setminus B) & \text{if } \sigma \in \Sigma_c \\ \varnothing & \text{otherwise} \end{cases} \tag{2}$$

The controller $\mathcal{C}$ is defined by:

$$\mathcal{C} = \langle \mathcal{S}, E \rangle \tag{3}$$

with the elements $\mathcal{S}$ and E defined as follows:

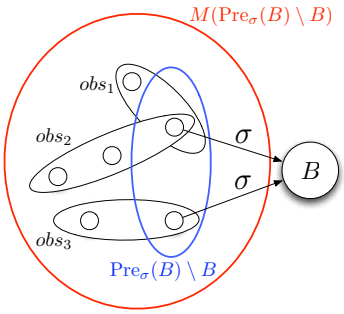

Fig. 4: An illustration of the Computation of $M(\mathrm{Pre}_\sigma(B) \setminus B)$.

1. the supervisory function $\mathcal{S}$ is given, for each observation $obs \in \mathcal{D}_{Obs}$, by
$$\mathcal{S}(obs) = \{\sigma \in \Sigma \,|\, obs \in \mathcal{F}(\sigma, I(Bad))\} \tag{4}$$
2. the set $E = I(Bad)$.

The computation of the function $\mathcal{F}$ is performed offline and, given an observation obs, the set $\mathcal{S}(obs)$ is computed online with the supervisory function $\mathcal{S}$ (defined in (4)), which uses the function $\mathcal{F}$. Since Σ is finite, $\mathcal{S}(obs)$ is computable when $I(Bad)$ is computed.

Proposition 5 *The controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, defined by (3), solves the basic problem.*

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

- *Base case* ($n = 0$): the projection on the initial states of the controlled system $\mathcal{T}_{/\mathcal{C}}$ is given by $\mathsf{Proj}_{\mathcal{D}_V}(\Theta_{/\mathcal{C}}) = \Theta \setminus E = \Theta \setminus I(Bad)$. Thus, the execution of $\mathcal{T}_{/\mathcal{C}}$ starts in a state that does not belong to $I(Bad)$.
- *Induction step*: suppose the proposition holds for paths of transitions of length less than or equal to n. We prove that this property remains true for paths of transitions of length $n+1$. By induction hypothesis, each state $\boldsymbol{\nu}$ reachable with a path of length n does not belong to $I(Bad)$. We show that no transition $\delta \in \Delta$ can be fired from this state $\boldsymbol{\nu} \notin I(Bad)$ to a state $\boldsymbol{\nu}' \in I(Bad)$. Indeed, either $\delta \in \Delta_c$, then this transition cannot be fired since $\sigma_\delta \in \mathcal{S}(obs)(\forall obs \in M(\boldsymbol{\nu}))$ by (2) and (3), or $\delta \in \Delta_{uc}$, then $\boldsymbol{\nu} \in I(Bad)$ (by (1)), which is impossible by hypothesis. □

Example 3 *For the STS of Fig. 1 and the mask of Example 2, the controller must ensure that, when the system is ready to consume some pieces of kind X (resp. X'), there are strictly more than ten pieces of this kind: $Bad = \{\langle \mathsf{CX}, x, x', y, y' \rangle | (x \leq 10) \wedge (y \in [0,2])\} \cup \{\langle \mathsf{CX}', x, x', y, y' \rangle | (x' \leq 10) \wedge (y' \in [0,2])\}$. The controllable (resp. uncontrollable) transitions are those drawn in plain (resp. dashed) lines. Then, the set $I(Bad)$ is defined as follows:*

$$\begin{aligned} I(Bad) = \; & Bad \cup \{\langle \mathsf{CX}, x, x', y, y' \rangle | [(x \leq 11) \wedge (y \in [1,2])] \vee [(x \leq 12) \wedge (y = 2)]\} \\ & \cup \{\langle \mathsf{CX}', x, x', y, y' \rangle | [(x' \leq 11) \wedge (y' \in [1,2])] \vee [(x' \leq 12) \wedge (y' = 2)]\} \end{aligned}$$

The computation of $\mathcal{F}$ gives:

$$\mathcal{F}(\sigma, I(Bad)) = \begin{cases} M(\{\langle \mathsf{PX}, x, x', y, y' \rangle | x \leq 12\}) & \\ \quad = \{\langle \mathsf{PX}, x, x', y, y' \rangle | x \leq 13\} & \text{if } \sigma = Stop\_prod \\ M(\{\langle \mathsf{PX}', x, x', y, y' \rangle | x' \leq 12\}) & \\ \quad = \{\langle \mathsf{PX}', x, x', y, y' \rangle | x' \leq 12\} & \text{if } \sigma = Stop\_prod' \\ \varnothing & \text{otherwise} \end{cases}$$

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

$$\mathcal{S}(obs) = \begin{cases} \{Stop\_prod\} & \text{if } obs \in \{\langle \mathsf{PX}, x, x', y, y' \rangle | x \leq 13\} \\ \{Stop\_prod'\} & \text{if } obs \in \{\langle \mathsf{PX}', x, x', y, y' \rangle | x' \leq 12\} \\ \varnothing & \text{otherwise} \end{cases}$$

One can note that the controller defined by this supervisory function $\mathcal{S}$ and by $I(Bad)$ is maximal. ◇

4.2 Effective Computation by Means of Abstract Interpretation

As seen in the previous section, the actual 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. Cousot and Cousot (1977); Halbwachs et al (1997); Jeannet (2003)) to compute an overapproximation of the fixpoint $I(Bad)$. This overapproximation ensures that the forbidden states Bad are not reachable in the controlled system. Thus, the synthesized controllers remain correct, but they can be less permissive.

Outline of the abstract interpretation techniques. In our case, 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 F is a monotonic function. We want to compute the least fixpoint (lfp) of a monotonic function $F : 2^{\mathcal{D}_V} \mapsto 2^{\mathcal{D}_V}$. Since $2^{\mathcal{D}_V}$ is a complete lattice, Tarski's theorem (Tarski (1955)) sentences that $\text{lfp}(F) = \cap\{c \in 2^{\mathcal{D}_V} \mid c \supseteq F(c)\}$. So, any post fixpoint c (with $c \supseteq F(c)$) is an overapproximation of $\text{lfp}(F)$.

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 (*possibly infinite*) abstract domain Λ, both domains having lattice structure. The concrete lattice $\langle 2^{\mathcal{D}_V}, \subseteq, \cup, \cap, \varnothing, \mathcal{D}_V \rangle$ and the abstract lattice $\langle \Lambda, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$ are linked by a Galois connection $2^{\mathcal{D}_V} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$, which ensures the correctness of the method as explained by Cousot and Cousot (1977).
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 ∇ 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 overapproximation of the least fixpoint of the function F.

For our experiments, we chose the abstract lattice of *convex polyhedra* (Cousot and Halbwachs (1978))[8]. A convex polyhedron on the n-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 right-angle triangle. In this lattice, $\sqcap$ is the classical intersection, $\sqcup$ is the convex hull and $\sqsubseteq$ is the inclusion. The widening operator (Cousot and Halbwachs (1978)) $P_1 \nabla P_2$ roughly consists in removing from P_1 all the constraints not satisfied by P_2. In other words, its principle is that 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. The *concretization function* $\gamma : \Lambda \mapsto 2^{\mathcal{D}_V}$ is defined by the identity function, whereas the *abstraction function* $\alpha : 2^{\mathcal{D}_V} \mapsto \Lambda$ is defined as follows: for each set $B \in 2^{\mathcal{D}_V}$, if this set corresponds to a polyhedron, then $\alpha(B)$ is defined by the least convex polyhedron which contains B, otherwise $\alpha(B)$ is defined by $\top$.

We assume in the sequel that the abstract lattice $\langle \Lambda, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$, the functions $\alpha : 2^{\mathcal{D}_V} \mapsto \Lambda$, $\gamma : \Lambda \mapsto 2^{\mathcal{D}_V}$ and the widening operator $\nabla : \Lambda \mapsto \Lambda$ are defined, with $2^{\mathcal{D}_V} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$.

Computation of the controller using abstract interpretation. The effective algorithm works like the semi-algorithm defined in section 4.1 except that we compute an overapproximation of $I(Bad)$ to ensure the termination. To compute this overapproximation, we first consider, in the abstract lattice, the function $\text{Pre}^\sharp_D : \Lambda \mapsto \Lambda$ (with $D \subseteq \Delta$), which corresponding to $\text{Pre}_D : 2^{\mathcal{D}_V} \mapsto 2^{\mathcal{D}_V}$ in the concrete one. It is defined, for each $l \in \Lambda$, by:

$$\text{Pre}^\sharp_D(l) = \bigsqcup_{\delta \in D} \text{Pre}^\sharp_\delta(l) \text{ , where } \text{Pre}^\sharp_\delta(l) = \alpha(G_\delta \cap A_\delta^{-1}(\gamma(l))) \tag{5}$$

Finally, $\text{Coreach}^\sharp_{uc}(Bad)$ is the least fixpoint of the function:

$$\lambda l.\alpha(Bad) \sqcup \text{Pre}^\sharp_{\Delta_{uc}}(l) \tag{6}$$

and we compute l_∞, defined as the limit of the sequence defined by $l_1 = \alpha(Bad)$ and $l_{i+1} = l_i \nabla \text{Pre}^\sharp_{uc}(l_i)$ (for $i > 1$). The abstract interpretation theory ensures that this sequence stabilizes after a finite number

[8] One can also consider interval (Cousot and Cousot (1977)) or octagons (Miné (2001)) abstract lattices. Note that Smacs (see section 7) implements these three lattices

of steps, and that $\gamma(l_\infty)$ is an overapproximation of $I(Bad)$ (recall that the fixpoint $I(Bad)$ always exists, but may be uncomputable). So, we obtain $I'(Bad) = \gamma(l_\infty) \supseteq I(Bad)$. Finally, we define the controller as in section 4.1 using $I'(Bad)$ instead of $I(Bad)$ and this controller is valid, because $I'(Bad) \supseteq I(Bad)$.

Example 4 *We consider a system with two variables x and y. Let us assume that $Bad = \{x = 0 \wedge y \geq 0\}$ and that there is an uncontrollable loop whose guard is $x \leq y$ and whose assignment is $x := x - 1$.*

So, $\mathrm{Pre}_{\Delta_{uc}}(Bad) = \{x = 0 \wedge y \geq 0\} \cup \{x = 1 \wedge y \geq 1\}$. Since we can iterate this loop, we have, after n computation steps, that $\mathrm{Coreach}_{uc}(Bad) = \{x = 0 \wedge y = 0\} \cup \{x = 1 \wedge y \geq 1\} \cup \cdots \cup \{x = n \wedge y \geq n\}$.

However, after only two computation steps, applying the widening operator gives the polyhedron: $I'(Bad) = \{0 \leq x \leq y\}$. In this example, there is no overapproximation, because each computation step can be represented exactly by a convex polyhedron. $\diamond$

Quality of the approximations. The method presented here always computes a safe controller, but without any guarantee that this controller is a maximal one. The less approximation we make during the computation, the more precise approximation of $I(Bad)$ we obtain. 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 overapproximations 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 given by Cousot and Halbwachs (1978) 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 (Halbwachs et al (1997)), we can use one of the fixpoint computation strategies defined by Bourdoncle (1992) and we can refine our abstract lattice (see Jeannet (2003) for more details).

There are however few theoretical results on the quality of the abstraction. We can only show, on empirical experiments (in section 7), that our abstractions allow the computation of interesting controllers.

In short, the effective computation of the memoryless controller can be summarized as follows. We compute an overapproximation $I'(Bad)$ of $I(Bad)$ to ensure the termination of the computation of the fixpoint. Then, we compute exactly the function $\mathcal{F}(\sigma, I'(Bad))$ $(\forall \sigma \in \Sigma)$ using (2). Finally, during the execution of the system, when the controller receives an observation obs, the value $\mathcal{S}(obs)$ is computed exactly using this function $\mathcal{F}$ in (4).

4.3 Evaluation of the Permissiveness of the Controller

In this section, we want to evaluate the quality in terms of permissiveness of our controller. For that, we compare it with the best memoryless controller that we know in the literature.

Takai and Kodama (1998) define a controller which, to our knowledge, is the most permissive controller known in the literature satisfying the *S-observability* condition (i.e. if $\boldsymbol{\nu}$ and $\boldsymbol{\nu}'$ have the same observation, then $\mathcal{S}$ will have the same control decision for both states[9]). However, this algorithm is only defined for finite LTS and for masks that are partitions of the state space (i.e. each state has one and only one observation). To our knowledge, there is no control algorithm defined for infinite systems with partial observation which explains our restriction to the finite case for the comparison.

Let us show that :

Proposition 6 *For finite systems and for masks that are partitions, our memoryless controller solving the basic problem and the one defined by Takai and Kodama (1998) have the same permissiveness.*

Proof Let us first explain the method given by Takai and Kodama (1998). The system to be controlled 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 $\delta : \Sigma \times X \mapsto X$ is the deterministic 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 $M : X \mapsto Y$, where Y is the finite observation space. The two steps of the algorithm of Takai and Kodama (1998) are:

[9] In our algorithm, the S-observability condition holds trivially, because the supervisory function is defined on the observation states.

1. the computation of the set of safe states $Q^{\uparrow} \subseteq Q$. $Q^{\uparrow} = \bigcap_{j=0}^{\infty} Q_j$, where Q_j is recursively defined as follows:

$$Q_j = \begin{cases} Q & \text{if } j = 0 \\ g(Q_{j-1}) & \text{otherwise} \end{cases}$$

   where, for each $Q' \subseteq Q$, the function $g(Q') = Q \cap (\bigcap_{\sigma \in \Sigma_{uc}} \{x \in X | (\langle \sigma, x \rangle \in \delta) \Rightarrow \delta(\sigma, x) \in Q'\})$.
2. the computation of the control function f, which gives the actions that are allowed by the controller. This function is defined for each $x \in Q^{\uparrow}$ by $f(x) = \{\sigma \in \Sigma_c | \exists x' \in Q^{\uparrow} : (M(x) = M(x')) \wedge (\langle \sigma, x' \rangle \in \delta) \wedge (\delta(\sigma, x') \in Q^{\uparrow})\}$.

Note that since the system is finite and the function g is monotonic, all the computations terminate in a finite amount of time; in particular, there is an n such that $Q^{\uparrow} = \bigcap_{j=0}^{n} Q_j$.

We remark that $Q^{\uparrow} = \overline{\text{Coreach}_{uc}(Bad)}$, because $\forall j \geq 0, \bigcap_{i \leq j} Q_i = \overline{\bigcup_{i \leq j} (\text{Pre}_{uc})^i (Bad)}$, where $(\text{Pre}_{uc})^0(Bad) = Bad$ and $(\text{Pre}_{uc})^i(Bad) = \text{Pre}_{uc}((\text{Pre}_{uc})^{i-1}(Bad))$ $(\forall i > 0)$.

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

5 The Deadlock Free Case

So far we have been interested in computing a controller solving the basic problem. But a solution to this problem does not ensure that the controlled system is deadlock free. Since most of the time, a deadlocking system cannot be tolerated, we extend in this section our method to synthesize a controller which solves the deadlock free problem. The general idea of the control is to compute, using fixpoint computation, the set $I_{bl}(Bad)$ of states that can lead to Bad or to deadlocking states triggering only uncontrollable transitions. Then, based on this set of states, we compute the controller.

5.1 Computation of the deadlock states

Computing $I_{bl}(Bad)$ requires to compute the states that would be in deadlock after control. So, we have to define the function $\text{Pre}_{bl}(B)$ which computes, for a set $B \subseteq \mathcal{D}_V$ of states to be forbidden, the set of states, that would be in deadlock in the controlled system, if the states of B were no longer reachable.

A state $\boldsymbol{\nu} \in \mathcal{D}_V$ will be in deadlock in the system $\mathcal{T}$ under the control of a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ that avoids B, if the two following conditions are satisfied in the system $\mathcal{T}$:

1. the state $\boldsymbol{\nu}$ has no outgoing uncontrollable transition.
2. there exists an observation $obs \in M(\boldsymbol{\nu})$ such that, for all controllable transitions δ, this transition cannot be fired from $\boldsymbol{\nu}$ or the action σ_δ is forbidden by control for this observation obs, i.e. $\sigma_\delta \in \mathcal{S}(obs)$, which is equivalent to $obs \in \mathcal{F}(\sigma_\delta, B)$.

Fig.5 illustrates the definition of deadlock states. The controllable action σ_1 is forbidden in $\boldsymbol{\nu}_1$, because a transition labelled by this action leads to B from this state, but the transition labelled by the uncontrollable action σ_3 can always be fired from $\boldsymbol{\nu}_1$, which implies that this state is deadlock free. The controllable action σ_2 is forbidden in $\boldsymbol{\nu}_2$, because a transition labelled by this action leads to B from this state and the controllable action σ_1 is forbidden in $\boldsymbol{\nu}_2$, because $\boldsymbol{\nu}_1$ and $\boldsymbol{\nu}_2$ are compatible with the observation obs_1. It implies that $\boldsymbol{\nu}_2$ is in deadlock. This can be formally defined by:

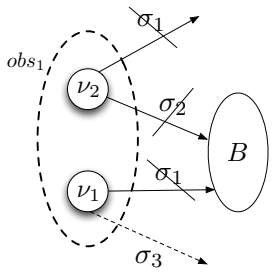

Fig. 5: Illustration of a deadlock state

Definition 7 *For a set of states $B \subseteq \mathcal{D}_V$ to be forbidden, a state $\boldsymbol{\nu}$ is in deadlock whenever:*

1. $\forall \delta \in \Delta_{uc} : \boldsymbol{\nu} \notin G_\delta$

2. $\exists obs \in M(\boldsymbol{\nu}), \forall \delta \in \Delta_c : (\boldsymbol{\nu} \notin G_\delta) \vee (obs \in \mathcal{F}(\sigma_\delta, B))$. •

Unfortunately, deciding for all states of the system if they are or not in deadlock is not computable with our symbolic operations. Indeed, for the second condition, we have to compute the intersection, for each transition $\delta \in \Delta_c$, of the union of $\overline{G_\delta}$ and $\mathcal{F}(\sigma_\delta, B)$. So, we have to combine, for each transition δ, the set of states $\overline{G_\delta}$ and the set of observations $\mathcal{F}(\sigma_\delta, B)$, but the domains of these sets is different. If we transform $\mathcal{F}(\sigma_\delta, B)$ into $M^{-1}(\mathcal{F}(\sigma_\delta, B))$, we lose the information about the observations and we need this information, because we have an existential quantification on the observations in this condition. If we transform $\overline{G_\delta}$ into $M(\overline{G_\delta})$, there can be states in $M^{-1}(M(\overline{G_\delta}))$ from which δ can be fired.

For this reason, we consider two cases, in which the problem can be solved:

1. In the first case, the mask M is a partition of $\mathcal{D}_V$. As a consequence, each state $\boldsymbol{\nu} \in \mathcal{D}_V$ is compatible with exactly one observation, which allows to simplify the definition of blocking states as follows:

   Definition 8 *For a set of states $B \subseteq \mathcal{D}_V$ to be forbidden, a state $\boldsymbol{\nu}$ is in deadlock whenever:*
   (a) $\forall \delta \in \Delta_{uc} : \boldsymbol{\nu} \notin G_\delta$
   (b) $\forall \delta \in \Delta_c : (\boldsymbol{\nu} \notin G_\delta) \vee (\boldsymbol{\nu} \in M^{-1}(\mathcal{F}(\sigma_\delta, B)))$. •

   A state $\boldsymbol{\nu}$ is in deadlock if it has no outgoing uncontrollable transition and if its outgoing controllable transitions δ are forbidden by control in the observation state $M(\boldsymbol{\nu})$ (i.e. $\sigma_\delta \in \mathcal{S}(M(\boldsymbol{\nu}))$ which is equivalent to $\boldsymbol{\nu} \in M^{-1}(\mathcal{F}(\sigma_\delta, B))$).
   Because $\forall \sigma \in \Sigma_{uc} : \mathcal{F}(\sigma, B) = \varnothing$, the function $\mathrm{Pre}_{bl}(B)$ (for $B \subseteq \mathcal{D}_V$), which gives the set of states, that would be in deadlock in the controlled system, if the states of B were no longer reachable, can be expressed as follows:

$$\mathrm{Pre}_{bl}(B) = B \cup \left[\bigcap_{\delta \in \Delta} \left(\overline{G_\delta} \cup (M^{-1}(\mathcal{F}(\sigma_\delta, B))) \right) \right] \tag{7}$$

2. In the second case, the mask M is a covering, but the observation space $\mathcal{D}_{Obs}$ is finite. We first define the function $L : \mathcal{D}_{Obs} \times \Sigma \times 2^{\mathcal{D}_V} \mapsto 2^{\mathcal{D}_V}$ as follows:

$$L(obs, \sigma, B) = \begin{cases} M^{-1}(obs) & \text{if } obs \in \mathcal{F}(\sigma, B) \\ \varnothing & \text{otherwise} \end{cases}$$

   For an observation state $obs \in \mathcal{D}_{Obs}$, an action $\sigma \in \Sigma$, and a set $B \subseteq \mathcal{D}_V$ of forbidden states, $L(obs, \sigma, B)$ gives the states compatible with obs if σ is forbidden by control in obs and the empty set otherwise.
   The idea to compute the deadlock states (for a set $B \subseteq \mathcal{D}_V$ of forbidden states) is to determine for each observation $obs \in \mathcal{D}_{Obs}$, the states $\boldsymbol{\nu}$:
   - that are compatible with obs (i.e. $\boldsymbol{\nu} \in M^{-1}(obs)$), and
   - that will be in deadlock for this observation if B is no longer reachable, i.e. for each transition δ, this transition cannot be fired from $\boldsymbol{\nu}$ (i.e. $\boldsymbol{\nu} \notin G_\delta$) or σ_δ is forbidden by control in the observation state obs of $\boldsymbol{\nu}$ (i.e. $\boldsymbol{\nu} \in L(obs, \sigma, B)$)

   The function Pre_{bl} can then be expressed as follows:

$$\mathrm{Pre}_{bl}(B) = B \cup \left[\bigcup_{obs \in \mathcal{D}_{Obs}} (M^{-1}(obs)) \cap \left(\bigcap_{\delta \in \Delta} (\overline{G_\delta} \cup (L(obs, \sigma_\delta, B)) \right) \right] \tag{8}$$

5.2 Symbolic computation of a deadlock free controlled system

Let us formalize the two steps of the construction (which uses $\mathrm{Pre}_{bl}(B)$):

Computation of $I_{bl}(Bad)$. This set of states and more generally $I_{bl}(.)$ is given by the function $\mathrm{Coreach}_{uc}^{bl} : 2^{\mathcal{D}_V} \mapsto 2^{\mathcal{D}_V}$ defined below. This set corresponds to the set of states that would be in deadlock in the controlled system or that lead to a forbidden state firing only uncontrollable transitions.

To compute $\mathrm{Coreach}_{uc}^{bl}(Bad)$, we first compute $\mathrm{Coreach}_{uc}(Bad)$ (defined by (1)). Then, if we make unreachable the forbidden states by cutting all the controllable transitions that lead to a bad state, the corresponding controlled system could have new deadlock states. We must add these deadlock states to

the set of forbidden states. The function $\mathrm{Pre}_{bl}(B)$ computes, for a set $B \subseteq \mathcal{D}_V$ of states to be forbidden, the set of states, that would be in deadlock in the controlled system, if the states of B were no longer reachable. The computation of the deadlock states is based on the function $\mathcal{F}$ defined at (2). To ensure the convergence in the computation of $\mathrm{Coreach}_{uc}^{bl}(Bad)$, Pre_{bl}, and therefore $\mathcal{F}$, must be monotonic. Thus, we use the monotonic function $\widehat{\mathcal{F}}$ instead of $\mathcal{F}$ in (7) and (8):

$$\widehat{\mathcal{F}}(\sigma, B) = \begin{cases} M(\mathrm{Pre}_\sigma(B)) & \text{if } \sigma \in \Sigma_c \\ \varnothing & \text{otherwise} \end{cases}$$

Remark 1 *Note that $\widehat{\mathcal{F}}$ is more restrictive than $\mathcal{F}$ and thus if we succeed in computing a controller w.r.t. $\mathcal{F}$, it is more permissive than a controller computed w.r.t. $\widehat{\mathcal{F}}$.* $\diamond$

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

$$\mathrm{Coreach}_{uc}^{bl}(Bad) = \mathrm{lfp}(\lambda B.Bad \cup \mathrm{Pre}_{bl}(\mathrm{Coreach}_{uc}(B))) \tag{9}$$

Computation of the controller $\mathcal{C}$. The controller $\mathcal{C}$ is defined similarly to what is done at section 4.1 using $I_{bl}(Bad)$ instead of $I(Bad)$, i.e. we compute offline the function $\mathcal{F}(\sigma, I_{bl}(Bad))$ $(\forall \sigma \in \Sigma)$ and online the function $\mathcal{S}(obs)$ (for each observation obs that the controller received from the system).

Proposition 7 *The controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, computed by the above algorithm, solves the deadlock free problem.*

Proof Since $\mathrm{Coreach}_{uc}(Bad) \subseteq \mathrm{Coreach}_{uc}^{bl}(Bad)$, it can be proved in a similar way as done in the proof of Prop. 5 that Bad is not reachable in this more restrictive controlled system.

Let us suppose that the controlled system does not satisfy the deadlock free property. Then, there exists at least a deadlock state $\boldsymbol{\nu} \in \mathcal{D}_V$, which is reachable in the controlled system. By definition of the fixpoint (9), $\boldsymbol{\nu} \in \mathrm{Coreach}_{uc}^{bl}(Bad)$, 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 the above algorithm, $\boldsymbol{\nu}$ and $\boldsymbol{\nu'}$ are both unreachable. $\square$

5.3 Effective computation by means of abstract interpretation

In order to compute the fixpoint $I_{bl}(Bad)$, we proceed similarly to what is done in section 4.2. For that, we transpose into the abstract lattice the function I_{bl} (denoted by $I_{bl}^\sharp$), which requires to do this transposition for the function $\mathrm{Coreach}_{uc}$ and Pre_{bl}. Formally, $I_{bl}^\sharp(Bad)$ is then the least fixpoint of the function:

$$\lambda \ell.\alpha(Bad) \sqcup \mathrm{Pre}_{bl}^\sharp(\mathrm{Coreach}_{uc}^\sharp(\ell))$$

where (i) the function $\mathrm{Coreach}_{uc}^\sharp(\ell)$ is defined by (6) and (ii) the function $\mathrm{Pre}_{bl}^\sharp : \Lambda \mapsto \Lambda$ (as a reminder, Λ is the abstract lattice) is defined, for each $\ell \in \Lambda$, by:

$$\mathrm{Pre}_{bl}^\sharp(\ell) \triangleq \ell \sqcup \left[\bigsqcap_{\langle \sigma, G, A \rangle \in \Delta} \left[\alpha(\overline{G}) \sqcup \left(\alpha(M^{-1}(M(\gamma(\mathrm{Pre}_{\mathrm{Trans}(\sigma)}^\sharp(\ell))))) \right) \right] \right]$$

with $\mathrm{Pre}_{\mathrm{Trans}(\sigma)}^\sharp(\ell)$ defined by (5).

We further compute ℓ_∞, defined as the limit of the sequence defined by $\ell_1 = \alpha(Bad)$ and $\ell_{i+1} = \ell_i \nabla \mathrm{Pre}_{bl}^\sharp(\mathrm{Coreach}_{uc}^\sharp(\ell_i))$ (for $i > 1$). The abstract interpretation theory ensures that this sequence converges after a finite number of steps and that ℓ_∞ is an overapproximation of $I_{bl}^\sharp(Bad)$. So, we obtain $I'_{bl}(Bad) = \gamma(\ell_\infty) \supseteq I_{bl}(Bad)$. One can note that $I_{bl}(Bad)$ is a nested fixpoint, because $\mathrm{Coreach}_{uc}$ is also defined by a fixpoint equation.

After the computation of $I'_{bl}(Bad)$, we compute exactly the function $\mathcal{F}(\sigma, I'_{bl}(Bad))$ $(\forall \sigma \in \Sigma)$. Finally, during the execution of the system, when the controller receives an observation obs, the value $\mathcal{S}(obs)$ is computed exactly using this function $\mathcal{F}$ in (4).

6 Improving the Permissiveness of the Controllers for Infinite Systems

In the previous sections, we provided methods to synthesize (non-blocking) memoryless controllers for infinite state systems under partial observation. In these algorithms, the control decisions taken by the controller are only based on the current observation that it receives from the system. When this method in not precise enough to give a satisfactory control, a way to improve the permissiveness of the controllers is to provide them with memory in order to retain partially or totally the history of the execution of the system, This allows the controller to have a finer knowledge of the set of states in which the system can be. A first possibility is to record the last k observations received from the system that can be used to refine the possible path taken by the system. A second possibility is to synthesize the controller on-line. In this section we only present these two extensions for the basic control problem knowing that they can be easily extended to the deadlock-free case.

6.1 Controller with Memory

Let us show how adding memory to the controller can improve its permissiveness. We suppose that the controller can memorize the last k observation states in which the system was, and uses this information to choose the actions to be forbidden. Note that the memoryless controller is a particular case of the controller with memory and corresponds to the case where k is equal to 1 (the controller only has the current observation). Thus, the propositions in section 3 remain valid for the controllers with memory.

k-Memory Controller and Controlled System. The controller with memory is formally defined as follows:

Definition 9 (k-Memory Controller) *For a fixed $k \geq 1$, a k-memory controller for $\mathcal{T}$ is a pair $\mathcal{C}_{\mathsf{m}} = \langle \mathcal{S}_{\mathsf{m}}, E_{\mathsf{m}} \rangle$, where:*

- *$\mathcal{S}_{\mathsf{m}} : (\mathcal{D}_{Obs})^k \mapsto 2^{\Sigma_c}$ is a supervisory function which defines, for a k-tuple of observations $\langle obs_1, \ldots, obs_k \rangle \in (\mathcal{D}_{Obs})^k$, a set $\mathcal{S}_{\mathsf{m}}(\langle obs_1, \ldots, obs_k \rangle)$ of controllable actions that have to be forbidden in any state $\boldsymbol{\nu} \in \mathcal{D}_V$ such that $obs_k \in M(\boldsymbol{\nu})$, when the memory of the controller is $\langle obs_1, \ldots, obs_k \rangle$.*
- *$E_{\mathsf{m}} \subseteq \mathcal{D}_V$ is a set of states to be forbidden, which restricts the set of initial states.* •

Let $\langle obs_1, \ldots, obs_k \rangle$ be the values in the memory of the controller, the element obs_k corresponds to the last observation that the controller received from the system; the controller knows thus that the current state of $\mathcal{T}$ is in $M^{-1}(obs_k)$. Moreover, when the controller receives a new observation obs_{k+1}, its memory evolves from $\langle obs_1, \ldots, obs_k \rangle$ to $\langle obs_2, \ldots, obs_{k+1} \rangle$. One can note that at the beginning of the system's execution, the memory of the controller is initialized with the value $\langle \varnothing, \ldots, \varnothing \rangle$.

The controlled system is formalized by an STS with k additional variables, which provide the last k observations the controller got from the system.

Definition 10 (Controlled STS) *For a fixed $k \geq 1$ and a controller $\mathcal{C}_{\mathsf{m}} = \langle \mathcal{S}_{\mathsf{m}}, E_{\mathsf{m}} \rangle$, the system $\mathcal{T}$ controlled by $\mathcal{C}_{\mathsf{m}}$, is an STS including observations $\mathcal{T}_{/\mathcal{C}_{\mathsf{m}}} = \langle V_{/\mathcal{C}_{\mathsf{m}}}, \Theta_{/\mathcal{C}_{\mathsf{m}}}, \Sigma, \Delta_{/\mathcal{C}_{\mathsf{m}}} \rangle$, where:*

- *$V_{/\mathcal{C}_{\mathsf{m}}}$ is a tuple of variables, whose domain is $\mathcal{D}_V \times (\mathcal{D}_{Obs})^k$.*
- *$\Theta_{/\mathcal{C}_{\mathsf{m}}} = \{\langle \boldsymbol{\nu}, \langle \varnothing, \ldots, \varnothing, obs \rangle \rangle | (\boldsymbol{\nu} \in (\Theta \setminus E_{\mathsf{m}})) \wedge (obs \in M(\boldsymbol{\nu}))\}$ is the initial condition.*
- *$\Delta_{/\mathcal{C}_{\mathsf{m}}}$ is defined from Δ as follows: for each transition $\langle \sigma, G, A \rangle \in \Delta$, we define a new transition $\langle \sigma, G_{/\mathcal{C}_{\mathsf{m}}}, A_{/\mathcal{C}_{\mathsf{m}}} \rangle \overset{\mathbf{def}}{\in} \Delta_{/\mathcal{C}_{\mathsf{m}}}$, where*
  (i) $G_{/\mathcal{C}_{\mathsf{m}}}$ is defined by

$$G_{/\mathcal{C}_{\mathsf{m}}} \triangleq \{\langle \boldsymbol{\nu}, \langle obs_1, \ldots, obs_k \rangle \rangle \,|(\boldsymbol{\nu} \in G) \wedge (\sigma \notin \mathcal{S}_{\mathsf{m}}(\langle obs_1, \ldots, obs_k \rangle)) \wedge (obs_k \in M(\boldsymbol{\nu}))\}$$

  (ii) $A_{/\mathcal{C}_{\mathsf{m}}}$ is defined, for each $\boldsymbol{\nu} \in \mathcal{D}_V$ and for each tuple of observations $\langle obs_1, \ldots, obs_k \rangle \in (\mathcal{D}_{Obs})^{k-1} \times M(\boldsymbol{\nu})$, by

$$A_{/\mathcal{C}_{\mathsf{m}}}(\langle \boldsymbol{\nu}, \langle obs_1, \ldots, obs_k \rangle \rangle) \triangleq \{\langle \boldsymbol{\nu}', \langle obs_2, \ldots, obs_k, obs_{k+1} \rangle \rangle \,| \\ (\boldsymbol{\nu}' = A(\boldsymbol{\nu})) \wedge (obs_{k+1} \in M(\boldsymbol{\nu}'))\}.$$

This definition is a generalization of Def. 5 with k-tuples of observations. A state $\langle \boldsymbol{\nu}, \langle obs_1, \ldots, obs_k \rangle \rangle$ of the controlled system models the fact that the system is in a state $\boldsymbol{\nu}$ and that $obs_1, \ldots, obs_k$ were the k last observations that the controller received. The guards and the update functions of the system are modified to take into account the k-tuples of observations:

1. *Guards:* A transition δ (labelled by σ) can be fired from a state $\langle \boldsymbol{\nu}, \langle obs_1, \ldots, obs_k \rangle \rangle$ in the controlled system $\mathcal{T}_{/\mathcal{C}}$, if δ can be fired from $\boldsymbol{\nu}$ in $\mathcal{T}$, $\boldsymbol{\nu}$ is compatible with obs_k, and σ is not forbidden by control for $\langle obs_1, \ldots, obs_k \rangle$ ($\mathcal{O}_\sigma^k$ is the set of k-tuples observation states for which the function $\mathcal{S}_{\mathsf{m}}$ allows to fire σ).
2. *Update functions:* The states $\langle \boldsymbol{\nu}', \langle obs_2, \ldots, obs_k, obs_{k+1} \rangle \rangle$ are reachable from the state $\langle \boldsymbol{\nu}, \langle obs_1, \ldots, obs_k \rangle \rangle$ through the transition δ in the controlled system $\mathcal{T}_{/\mathcal{C}}$, if $\boldsymbol{\nu}'$ is reachable from $\boldsymbol{\nu}$ through δ in $\mathcal{T}$, and $\boldsymbol{\nu}'$ is compatible with obs_{k+1}. The definition of $A_{/\mathcal{C}_{\mathsf{m}}}$ is based on the evolution of the controller's memory when a new observation obs'; the memory evolves from $\langle obs_1, \ldots, obs_k \rangle$ to $\langle obs_2, \ldots, obs_k, obs_{k+1} \rangle$. Moreover, this function is such that if $\langle \boldsymbol{\nu}', \langle obs_2, \ldots, obs_k, obs' \rangle \rangle \in A_{/\mathcal{C}}(\langle \boldsymbol{\nu}, \langle obs_1, \ldots, obs_k \rangle \rangle)$, then $\langle \boldsymbol{\nu}', \langle obs_2, \ldots, obs_k, obs'_{k+1} \rangle \rangle \in A_{/\mathcal{C}}(\langle \boldsymbol{\nu}, \langle obs_1, \ldots, obs_k \rangle \rangle)$ for each $obs'_{k+1} \in M(\boldsymbol{\nu}')$. This allows to model the fact that when the system arrives in the state $\boldsymbol{\nu}'$, then the controller can receive any observation among those in $M(\boldsymbol{\nu}')$.

In these settings, we are interested in the problem:

Problem 3 (Memory Basic State Avoidance Control Problem) *The Memory Basic State Avoidance Control Problem (memory basic problem for short) is defined as the basic problem, but with a k-memory controller instead of a memoryless one.*

As we shall see, using a k-memory controller provides a finer control than the one that is given by the memoryless controller, since it has a more precise knowledge concerning the set of states in which the system can be.

Symbolic computation of the controller. We extend the algorithm presented in the memoryless case to include the memory. The construction consists in computing the set $I(Bad)$ of states that can lead to Bad triggering only uncontrollable transitions using the fixpoint given by (1) and then in computing the controller $\mathcal{C}_{\mathsf{m}}$ whose aim is to disable all the controllable actions that may lead to a state in $I(Bad)$. For this second step, we first define the function $\mathcal{F}_{\mathsf{m}} : \Sigma \times 2^{\mathcal{D}_V} \mapsto (2^{\mathcal{D}_{Obs}})^k$, where for an action $\sigma \in \Sigma$ and a set $B \subseteq \mathcal{D}_V$ of states to be forbidden, $\mathcal{F}_{\mathsf{m}}(\sigma, B)$ specifies a k-tuple of sets of observation states for which the action σ has to be forbidden. Each k-tuple $\langle obs_1, \ldots, obs_k \rangle \in \mathcal{F}_{\mathsf{m}}(\sigma, B)$ should correspond to a possible sequence of observations of the system ending with an observation obs_k such that there exists a state $\boldsymbol{\nu} \in M^{-1}(obs_k)$, from which a transition labelled by σ leads to B.

In order to compute $\mathcal{F}_{\mathsf{m}}(\sigma, B)$, we first compute the set $\mathcal{P}_\sigma^B$ of possible executions of the system ending with a state $\boldsymbol{\nu}$ leading to B through a transition labelled by σ and then we apply the mask M to them to take into account the paths that are indistinguishable from them. The set $\mathcal{P}_\sigma^B$ is defined for each $B \subseteq \mathcal{D}_V$ and $\sigma \in \Sigma$ as follows:

$$\mathcal{P}_\sigma^B = \langle (\mathrm{Pre}_\Sigma^B)^{k-1}(\mathrm{Pre}_\sigma(B) \setminus B), \ldots, (\mathrm{Pre}_\Sigma^B)^1(\mathrm{Pre}_\sigma(B) \setminus B), \mathrm{Pre}_\sigma(B) \setminus B \rangle$$

where $(\mathrm{Pre}_\Sigma^B)^i(B')$ is defined, for each $B, B' \subseteq \mathcal{D}_V$, in the following way:

$$(\mathrm{Pre}_\Sigma^B)^i(B') = \begin{cases} \mathrm{Pre}_\Sigma((\mathrm{Pre}_\Sigma^B)^{i-1}(B')) \setminus B & \text{if } i \geq 1 \\ B' & \text{if } i = 0 \end{cases}$$

The k^{th} element of $\mathcal{P}_\sigma^B$ is the set of states $\boldsymbol{\nu} \notin B$ leading to B through a transition labelled by σ and the i^{th} element ($\forall i \in [1, k-1]$) of $\mathcal{P}_\sigma^B$ is the set of states $\boldsymbol{\nu} \notin B$ leading to the $(i+1)^{th}$ element of $\mathcal{P}_\sigma^B$ through a transition labelled by any transition in Σ (see Fig. 6).

Remark 2 *$\mathcal{P}_\sigma^B$ is only an overapproximation of the paths ending with a state $\boldsymbol{\nu}$ leading to B through a transition labelled by σ. Indeed, if we have the paths $\boldsymbol{\nu}_1 \xrightarrow{\sigma_1} \boldsymbol{\nu}_2 \xrightarrow{\sigma} B$ and $\boldsymbol{\nu}_3 \xrightarrow{\sigma_2} \boldsymbol{\nu}_4 \xrightarrow{\sigma} B$, then the computation of $\mathcal{P}_\sigma^B$ for the paths of length 2 gives $\langle \{\boldsymbol{\nu}_1, \boldsymbol{\nu}_3\}, \{\boldsymbol{\nu}_2, \boldsymbol{\nu}_4\} \rangle$. The path of states $\boldsymbol{\nu}_1, \boldsymbol{\nu}_4$ is thus considered as a possible path, whereas it is not the case. An exact computation must record all the precise paths to B with a transition labelled by σ at the end. Unfortunately, with a infinite state space, such a precise computation will not always terminate.* ◇

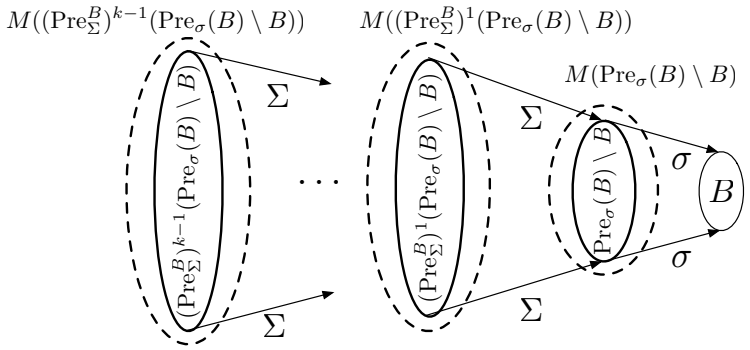

Fig. 6: The paths of length k leading to B.

In order to compute the function $\mathcal{F}_\mathsf{m}(\sigma, B)$, we simply apply the mask M to each element of $\mathcal{P}_\sigma^B$:

$$\mathcal{F}_\mathsf{m}(\sigma, B) = \begin{cases} \langle M((\mathrm{Pre}_\Sigma^B)^{k-1}(\mathrm{Pre}_\sigma(B) \setminus B)), \ldots, M((\mathrm{Pre}_\Sigma^B)^1(\mathrm{Pre}_\sigma(B) \setminus B)), \\ \ M(\mathrm{Pre}_\sigma(B) \setminus B)\rangle & \text{if } \sigma \in \Sigma_c \\ \varnothing & \text{otherwise} \end{cases} \tag{10}$$

The controller $\mathcal{C}_\mathsf{m}$ is given by:

$$\mathcal{C}_\mathsf{m} = \langle \mathcal{S}_\mathsf{m}, E_\mathsf{m} \rangle \tag{11}$$

with the element $\mathcal{S}_\mathsf{m}$ and E_m defined as follows:

1. the supervisory function $\mathcal{S}_\mathsf{m}$ is given, for each possible value $\langle obs_1, \ldots, obs_k \rangle$ in the memory of the controller, by:

$$\mathcal{S}_\mathsf{m}(\langle obs_1, \ldots, obs_k \rangle) = \{\sigma \in \Sigma \,|\, \langle obs_1, \ldots, obs_k \rangle \in \mathcal{F}_\mathsf{m}(\sigma, I(Bad))\} \tag{12}$$

2. the set $E_\mathsf{m} = I(Bad)$.

Proposition 8 *The controller $\mathcal{C}_\mathsf{m} = \langle \mathcal{S}_\mathsf{m}, E_\mathsf{m} \rangle$, defined by (11), solves the memory basic problem.*

The proof is similar to the one of Prop. 5.

Example 5 *This example illustrates how the computation of a k-memory controller works and shows that this one is more permissive than the memoryless controller computed for the same example. The STS of Fig. 7 has explicit locations $\ell \in \{\ell_1, \ell_2, \ell_3, \ell_b\}$ and a variable $x \in [1, \infty]$. For each guard in Fig. 7, the element t is initialized[10] with the value 2 and its action is controllable. The set of initial states is $\{\langle \ell_1, 1\rangle, \langle \ell_2, 1\rangle\}$ and the observer $\langle Obs, M\rangle$ is such that $M(\langle \ell_1, 1\rangle) = \{obs_1\}$, $M(\langle \ell_2, 1\rangle) = \{obs_1'\}$, $M(\langle \ell_1, i\rangle) = M(\langle \ell_2, i\rangle) = \{obs_i\}$ $(\forall i > 1)$, and $M(\langle \ell_3, i\rangle) = M(\langle \ell_b, i\rangle) = \{obs_b\}$ $(\forall i \geq 1)$. The set $Bad = \{\langle \ell_b, i\rangle | \, \forall i \in \mathbb{N}\}$ and we have that $I(Bad) = Bad$. We show that the k-memory controller allows the system to reach the location ℓ_3, whereas the memoryless controller forbids the access to ℓ_3.*

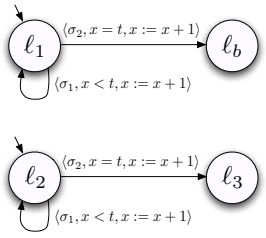

Fig. 7: Comparison between controllers.

– *Computation of $\mathcal{C}_\mathsf{m} = \langle \mathcal{S}_\mathsf{m}, I(Bad)\rangle$: we first compute $\mathcal{F}_\mathsf{m}(\sigma_2, I(Bad)) = \langle M(\langle \ell_1, 1\rangle), M(\langle \ell_1, 2\rangle)\rangle = \langle obs_1, obs_2\rangle$. Thus, $\mathcal{S}_\mathsf{m}$ is defined as follows:*

$$\mathcal{S}_\mathsf{m}(\langle obs, obs'\rangle) = \begin{cases} \{\sigma_2\} & \textit{if } (obs = obs_1) \wedge (obs' = obs_2) \\ \varnothing & \textit{otherwise} \end{cases}$$

The action σ_2 is not forbidden in the state $\langle \ell_2, 2\rangle$, if the controller $\mathcal{C}_\mathsf{m}$ has observed obs_1' before obs_2.

[10] We shall reuse this example with a different value for t further in the paper.

– *Computation of* $\mathcal{C} = \langle \mathcal{S}, I(bad)\rangle$ *: we compute* $\mathcal{F}(\sigma_2, I(Bad)) = M(\langle \ell_1, 2\rangle) = \{obs_2\}$*. Thus,* $\mathcal{S}$ *is defined as follows:*

$$\mathcal{S}(obs) = \begin{cases} \{\sigma_2\} & \text{if } obs = obs_1 \\ \varnothing & \text{otherwise} \end{cases}$$

The action σ_2 *is always forbidden in the state* $\langle \ell_2, 2\rangle$ *even if the controller* $\mathcal{C}$ *has observed* obs'_1 *before* obs_2*.*

It is thus easy to see that the controller $\mathcal{C}_\mathsf{m}$ *is more permissive than the controller* $\mathcal{C}$ *on this example.* $\diamond$

Now, we prove that a k'-memory controller is more permissive than a k-memory controller when $k' > k$.

Proposition 9 *For any* $k' > k \geq 1$*, the* k'*-memory controller* $\mathcal{C}'_\mathsf{m} = \langle \mathcal{S}'_\mathsf{m}, E'_\mathsf{m}\rangle$ *is more permissive than the* k*-memory controller* $\mathcal{C}_\mathsf{m} = \langle \mathcal{S}_\mathsf{m}, E_\mathsf{m}\rangle$*.*

Proof We use I to denote $I(Bad)$. We prove that for all tuples of observations $\langle obs_1, \ldots, obs_{k'}\rangle$ in $(\mathcal{D}_{Obs} \cup \{\varnothing\})^{k'}$, $\forall \sigma \in \Sigma_c$: $\sigma \in \mathcal{S}'_\mathsf{m}(\langle obs_1, \ldots, obs_{k'}\rangle) \Rightarrow \sigma \in \mathcal{S}_\mathsf{m}(\langle obs_{k'-k+1}, \ldots, obs_{k'}\rangle)$:

$$\begin{aligned}
&\sigma \in \mathcal{S}'_\mathsf{m}(\langle obs_1, \ldots, obs_{k'}\rangle) \\
\Rightarrow\ &\langle obs_1, \ldots, obs_{k'}\rangle \in \mathcal{F}'_\mathsf{m}(\sigma, I) \\
\Rightarrow\ &\langle obs_1, \ldots, obs_{k'}\rangle \in \langle M((\mathrm{Pre}^I_\Sigma)^{k'-1}(\mathrm{Pre}_\sigma(I) \setminus I)), \ldots, \\
&\qquad\qquad M((\mathrm{Pre}^I_\Sigma)^1(\mathrm{Pre}_\sigma(I) \setminus I)), M(\mathrm{Pre}_\sigma(I) \setminus I)\rangle \\
\Rightarrow\ &\langle obs_{k'-k+1}, \ldots, obs_{k'}\rangle \in \langle M((\mathrm{Pre}^I_\Sigma)^{k-1}(\mathrm{Pre}_\sigma(I) \setminus I)), \ldots, \\
&\qquad\qquad M((\mathrm{Pre}^I_\Sigma)^1(\mathrm{Pre}_\sigma(I) \setminus I)), M(\mathrm{Pre}_\sigma(I) \setminus I)\rangle \\
\Rightarrow\ &\langle obs_{k'-k+1}, \ldots, obs_{k'}\rangle \in \mathcal{F}_\mathsf{m}(\sigma, I), \text{ by definition of } \mathcal{F}_\mathsf{m} \\
\Rightarrow\ &\sigma \in \mathcal{S}_\mathsf{m}(\langle obs_{k'-k+1}, \ldots, obs_{k'}\rangle)
\end{aligned}$$

□

Effective Computation by Means of Abstract Interpretation. As in section 4.2, an effective algorithm can be obtained from the algorithm described above using abstract interpretation techniques and it can be summarized as follows. We compute an overapproximation $I'(Bad)$ of $I(Bad)$. Then, we compute exactly the function $\mathcal{F}_\mathsf{m}(\sigma, I'(Bad))$ $(\forall \sigma \in \Sigma)$ using (10). Finally, during the execution of the system, when the controller receives an observation, the actions to be forbidden given by $\mathcal{S}_\mathsf{m}$ are computed exactly using the function $\mathcal{F}_\mathsf{m}$ in (12).

In this section we have seen that the k-memory controller is more permissive than the memoryless controller. However, it requires additional offline computations to compute the function $\mathcal{F}_\mathsf{m}$. Increasing the size k of the memory of the controller gives a more permissive solution, but it also increases the time complexity to compute it (the time complexity to compute is k times greater than for the memoryless method). Similarly, for the online part of these two methods, the computation cost for the k-memory is k times greater than the one of the memoryless method. The choice between these two controllers thus depends on the requirement of the user. The k-memory controller must be used when the aim is to obtain a solution of better quality and the memoryless controller must be used when the purpose is to quickly compute a solution.

6.2 Online Controller

A further improvement of the controller consists in defining an online controller $\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_\mathsf{o}, E_\mathsf{o}\rangle$. The supervisory function $\mathcal{S}_\mathsf{o}$ uses all the information available corresponding to the past of the execution of the system to determine the actions to be forbidden. The supervisory function is a *dynamic* function, which evolves according to the execution of the system and we will denote by $\mathcal{S}_{\mathsf{o},i}$ the value of this function computed at the i^{th} step of the execution of the system. We will show that such an online controller is more precise that any k-memory offline controller. Our online controller is based of the one of Kumar et al (1993) which is defined only for finite systems.

Symbolic computation of the controller. The online algorithm is defined as follows. When the controller receives a new observation obs_i, we compute the possible current states P_i (defined further) according to this observation, the possible current states P_{i-1} in which the system could be at the previous step, and the controller $\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_{\mathsf{o},i-1}, E_\mathsf{o} \rangle$ computed at the previous step. Then, the supervisory function $\mathcal{S}_{\mathsf{o},i}$ is computed according to this set P_i and it forbids the actions which allow to reach $I(Bad)$ from P_i. The supervisory function $\mathcal{S}_\mathsf{o}$ is thus a mapping $2^{\mathcal{D}_V} \mapsto 2^{\Sigma_c}$. More precisely, the online algorithm is is defined as follows:

1. We compute the set $I(Bad)$ according to (1) and we define the controller $\mathcal{C}_\mathsf{o}$ as follows:
$$\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_\mathsf{o}, E_\mathsf{o} \rangle \tag{13}$$
where the supervisory function $\mathcal{S}_\mathsf{o}$ is defined in each step according to the observations received by the controller so far and the set $E_\mathsf{o} = I(Bad)$. The set of initial states in the controlled system $\mathcal{T}_{/\mathcal{C}_\mathsf{o}}$ is thus given by $\Theta_{/\mathcal{C}_\mathsf{o}} = \Theta \setminus E_\mathsf{o} = \Theta \setminus I(Bad)$.
2. Let obs_i be the i^{th} observation (for $i \geq 1$) that the controller got from the system, P_{i-1} be the possible states in which the system could be at the previous step, and $\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_{\mathsf{o},i-1}, E_\mathsf{o} \rangle$ be the controller computed at the previous step. The possible current states P_i are computed as follows:
$$P_i = \begin{cases} M^{-1}(obs_i) \cap \mathrm{Post}_{\Sigma \setminus \mathcal{S}_{\mathsf{o},i-1}(P_{i-1})}(P_{i-1}) & \text{if } i > 1 \\ \Theta_{/\mathcal{C}_\mathsf{o}} \cap M^{-1}(obs_1) & \text{if } i = 1 \end{cases} \tag{14}$$
where $\mathrm{Post}_{\Sigma \setminus \mathcal{S}_{\mathsf{o},i-1}(P_{i-1})}(P_{i-1})$ gives the states that the controller $\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_{\mathsf{o},i-1}, E_\mathsf{o} \rangle$ allows to reach from P_{i-1} firing exactly an allowed transition.
Then, the supervisory function $\mathcal{S}_{\mathsf{o},i}$ is defined with respect to P_i as follows:
$$\mathcal{S}_{\mathsf{o},i}(P_i) = \{\sigma \in \Sigma_c \mid \mathrm{Post}_\sigma(P_i) \cap I(Bad) \neq \varnothing\} \tag{15}$$
It means that σ is forbidden if $I(Bad)$ is reachable from P_i through a transition labelled by σ.

Proposition 10 *The online controller $\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_\mathsf{o}, E_\mathsf{o} \rangle$, defined by (13), solves the basic problem.*

The proof is similar to the one of Prop. 5 and is omitted.

Example 6 *This example illustrates how the computation of an online controller works and shows that this one is more permissive than the k-memory controller computed for the same example. We consider the STS of Fig. 7 in order to compare the online controller $\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_\mathsf{o}, E_\mathsf{o} \rangle$ with the k-memory controller $\mathcal{C}_\mathsf{m} = \langle \mathcal{S}_\mathsf{m}, E_\mathsf{m} \rangle$. For each guard in Fig. 7, the element t is initialized with the value $k+1$ and its action is controllable. The set of initial states, the observer $\langle Obs, M \rangle$, the set Bad and the set $I(Bad)$ are defined as in Example 5. We show that the online controller allows to reach the location ℓ_3, whereas the k-memory controller forbids the access to ℓ_3:*

- *Computation of $\mathcal{C}_\mathsf{m} = \langle \mathcal{S}_\mathsf{m}, I(Bad) \rangle$: We compute $\mathcal{F}_\mathsf{m}(\sigma_2, I(Bad)) = \langle M(\langle \ell_1, 2 \rangle), \ldots, M(\langle \ell_1, k+1 \rangle) \rangle = \langle obs_2, \ldots, obs_{k+1} \rangle$ and we have that $\mathcal{S}_\mathsf{m}$ is defined as follows:*
$$\mathcal{S}_\mathsf{m}(\langle o_1, \ldots, o_k \rangle) = \begin{cases} \{\sigma_2\} & \text{if } \forall i \in [1,k] : o_i = obs_{i+1} \\ \varnothing & \text{otherwise} \end{cases}$$
There are two possible executions in controlled system $\mathcal{T}_{/\mathcal{C}_\mathsf{m}}$:
  1. *the execution $\langle \ell_1, 1 \rangle, \langle \ell_1, 2 \rangle, \ldots, \langle \ell_1, k+1 \rangle$: when the system arrives in the state $\langle \ell_1, k+1 \rangle$, it cannot fire the action σ_2, because its memory is equal to $\langle obs_2, \ldots, obs_{k+1} \rangle$.*
  2. *the execution $\langle \ell_2, 1 \rangle, \langle \ell_2, 2 \rangle, \ldots, \langle \ell_2, k+1 \rangle$: when the system arrives in the state $\langle \ell_2, k+1 \rangle$, it cannot fire the action σ_2, because its memory is equal to $\langle obs_2, \ldots, obs_{k+1} \rangle$.*

  The location ℓ_3 is thus not reachable in the $\mathcal{T}_\mathsf{m}$.
- *Computation of the online controller: Let us consider the execution $\langle \ell_2, 1 \rangle, \langle \ell_2, 2 \rangle, \ldots, \langle \ell_2, k+1 \rangle$. At the beginning, the set $P_1 = (\Theta \setminus I(Bad)) \cap M^{-1}(obs'_1) = \{\langle \ell_1, 1 \rangle, \langle \ell_2, 1 \rangle\} \cap \{\langle \ell_2, 1 \rangle\} = \{\langle \ell_2, 1 \rangle\}$ and the supervisory function $\mathcal{S}_{\mathsf{o},1}$ forbids no action. For the next observation obs_2, the set $P_2 = \{\langle \ell_1, 2 \rangle, \langle \ell_2, 2 \rangle\} \cap \{\langle \ell_2, 2 \rangle\} = \{\langle \ell_2, 2 \rangle\}$ and the supervisory function $\mathcal{S}_{\mathsf{o},2}$ forbids no action. Similarly, for the observation obs_i ($\forall i \in [3,k]$), the set $P_i = \{\langle \ell_1, i \rangle, \langle \ell_2, i \rangle\} \cap \{\langle \ell_2, i \rangle\} = \{\langle \ell_2, i \rangle\}$ and the supervisory function $\mathcal{S}_{\mathsf{o},i}$ forbids no action. For the last observation obs_{k+1}, the set $P_{k+1} = \{\langle \ell_1, k+1 \rangle, \langle \ell_2, k+1 \rangle\} \cap \{\langle \ell_2, k+1 \rangle\} = \{\langle \ell_2, k+1 \rangle\}$ and the supervisory function $\mathcal{S}_{\mathsf{o},k+1}$ forbids no action. Thus, the location ℓ_3 is reachable from $\langle \ell_2, k+1 \rangle$ with the online controller. One can also verify that σ_2 is forbidden in the state $\langle \ell_1, k+1 \rangle$ after the path $\langle \ell_1, 1 \rangle, \langle \ell_1, 2 \rangle, \ldots, \langle \ell_1, k+1 \rangle$.*

It is easy to see that the online controller $\mathcal{C}_\mathsf{o}$ is more permissive than the k-memory controller $\mathcal{C}_\mathsf{m}$ on this example. $\diamond$

Next, we formally prove that the online controller is more permissive than the k-memory controller.

Proposition 11 *For any $k \geq 1$, the online controller $\mathcal{C}_\mathsf{o} = \langle \mathcal{S}_\mathsf{o}, E_\mathsf{o} \rangle$ is more permissive than the k-memory controller $\mathcal{C}_\mathsf{m} = \langle \mathcal{S}_\mathsf{m}, E_\mathsf{m} \rangle$.*

Proof In the proof, we use I to denote $I(Bad)$. Let $obs_1, \ldots, obs_j$ be a sequence of observation states the controller got from the system. By Prop. 10, we have for each $i \in [1, j]$ that:

$$P_i \cap I(Bad) = \varnothing \tag{16}$$

We prove that an action forbidden by the online controller is also forbidden by the k-memory controller. For that, we prove for each $\sigma \in \Sigma_c$ that if $\sigma \in \mathcal{S}_{\mathsf{o},j}(P_j)$, then $\sigma \in \mathcal{S}_\mathsf{m}(\langle obs_{j-k+1}, \ldots, obs_j \rangle)$ (if $j \geq k$) or $\sigma \in \mathcal{S}_\mathsf{m}(\langle \varnothing, \ldots, \varnothing, obs_1, \ldots, obs_j \rangle)$ (if $j < k$). Thus, we consider two cases:

1. *If $j \geq k$*: we first prove that $obs_j \in M(\mathrm{Pre}_\sigma(I) \setminus I)$.

$$\begin{aligned}
&\sigma \in \mathcal{S}_{\mathsf{o},j}(P_j) \\
\Rightarrow\ &\mathrm{Post}_\sigma(P_j) \cap I \neq \varnothing \\
\Rightarrow\ &\exists \boldsymbol{\nu}_1 \in P_j : \boldsymbol{\nu}_1 \in \mathrm{Pre}_\sigma(I) \\
\Rightarrow\ &\exists \boldsymbol{\nu}_1 \in P_j : \boldsymbol{\nu}_1 \in (\mathrm{Pre}_\sigma(I) \setminus I), \text{ because } \boldsymbol{\nu}_1 \notin I \text{ by (16)} \qquad (\alpha) \\
\Rightarrow\ &obs_j \in M(\mathrm{Pre}_\sigma(I) \setminus I), \text{ because } obs_j \in M(\boldsymbol{\nu}_1) \text{ by (14)}
\end{aligned}$$

   Then, we prove that $obs_{j-1} \in M((\mathrm{Pre}^I_\Sigma)^1(\mathrm{Pre}_\sigma(I) \setminus I))$. From (α), we know that $\exists \boldsymbol{\nu}_1 \in P_j : \boldsymbol{\nu}_1 \in \mathrm{Pre}_\sigma(I) \setminus I$. Moreover, for this state $\boldsymbol{\nu}_1$, we have:

$$\begin{aligned}
&\boldsymbol{\nu}_1 \in P_j \\
\Rightarrow\ &\boldsymbol{\nu}_1 \in \mathrm{Post}_{\Sigma \setminus \mathcal{S}_{j-1}(P_{j-1})}(P_{j-1}) \\
\Rightarrow\ &\exists \boldsymbol{\nu}_2 \in P_{j-1} : \boldsymbol{\nu}_1 \in \mathrm{Post}_{\Sigma \setminus \mathcal{S}_{j-1}(P_{j-1})}(\boldsymbol{\nu}_2) \\
\Rightarrow\ &\exists \boldsymbol{\nu}_2 \in P_{j-1} : \boldsymbol{\nu}_2 \in \mathrm{Pre}_\Sigma(\boldsymbol{\nu}_1) \\
\Rightarrow\ &\exists \boldsymbol{\nu}_2 \in P_{j-1} : \boldsymbol{\nu}_2 \in \mathrm{Pre}_\Sigma(\mathrm{Pre}_\sigma(I) \setminus I), \text{because } \boldsymbol{\nu}_1 \in (\mathrm{Pre}_\sigma(I) \setminus I) \text{ by } (\alpha) \\
\Rightarrow\ &\exists \boldsymbol{\nu}_2 \in P_{j-1} : \boldsymbol{\nu}_2 \in (\mathrm{Pre}_\Sigma(\mathrm{Pre}_\sigma(I) \setminus I) \setminus I), \text{because } \boldsymbol{\nu}_2 \notin I \text{ by (16)} \\
\Rightarrow\ &\exists \boldsymbol{\nu}_2 \in P_{j-1} : \boldsymbol{\nu}_2 \in (\mathrm{Pre}^I_\Sigma)^1(\mathrm{Pre}_\sigma(I) \setminus I) \qquad (\beta) \\
\Rightarrow\ &obs_{j-1} \in M((\mathrm{Pre}^I_\Sigma)^1(\mathrm{Pre}_\sigma(I) \setminus I)), \text{ because } obs_{j-1} \in M(\boldsymbol{\nu}_2) \text{ by (14)}
\end{aligned}$$

   From (β), we know that $\exists \boldsymbol{\nu}_2 \in P_{j-1} : \boldsymbol{\nu}_2 \in (\mathrm{Pre}^I_\Sigma)^1(\mathrm{Pre}_\sigma(I) \setminus I)$ and we can prove similarly as above that $obs_{j-2} \in M((\mathrm{Pre}^I_\Sigma)^2(\mathrm{Pre}_\sigma(I) \setminus I))$ and that $\exists \boldsymbol{\nu}_3 \in P_{j-2} : \boldsymbol{\nu}_3 \in (\mathrm{Pre}^I_\Sigma)^2(\mathrm{Pre}_\sigma(I) \setminus I)$. Then, with the same reasoning, we prove that $obs_{j-i} \in M((\mathrm{Pre}^I_\Sigma)^i(\mathrm{Pre}_\sigma(I) \setminus I))$ and that $\exists \boldsymbol{\nu}_{i+1} \in P_{j-i} : \boldsymbol{\nu}_{i+1} \in (\mathrm{Pre}^I_\Sigma)^i(\mathrm{Pre}_\sigma(I) \setminus I)$ for $i = 3, 4, \ldots, k-1$. Thus, we proved that $obs_{j-i} \in M((\mathrm{Pre}^I_\Sigma)^i(\mathrm{Pre}_\sigma(I) \setminus I))$ $(\forall i \in [1, k-1])$ and that $obs_j \in M(\mathrm{Pre}_\sigma(I) \setminus I)$. It implies that $\langle obs_{j-k+1}, \ldots, obs_j \rangle \in \mathcal{F}_\mathsf{m}(\sigma, I(Bad))$ by (10), and thus that $\sigma \in \mathcal{S}_\mathsf{m}(\langle obs_{j-k+1}, \ldots, obs_j \rangle)$.
2. *If $j < k$*: we can prove similarly that $\langle \varnothing, \ldots, \varnothing, obs_1, \ldots, obs_j \rangle \in \mathcal{F}_\mathsf{m}(\sigma, I(Bad))$. $\square$

Effective Computation by Means of Abstract Interpretation. As in section 4.2, an effective algorithm can be obtained from the algorithm described above using abstract interpretation techniques and it can be summarized as follows. We compute an overapproximation $I'(Bad)$ of $I(Bad)$. Then, during the execution of the system, when the controller receives a new observation obs_i, the values P_i and $\mathcal{S}_{\mathsf{o},i}(P_i)$ are respectively computed according to (14) and (15) using $I'(Bad)$ instead of $I(Bad)$.

In this section, we have seen that the online controller is more permissive than the k-memory and memoryless controllers. However, it requires significant online computations to choose the actions to be forbidden. Indeed even though the complexity of the offline part is similar to the memoryless method, the on-line part of the memoryless method only requires a membership test, whereas the on-line test requires some computations having a polynomial time complexity. In particular, if the time to choose the actions to be forbidden is short, we will not always be able to use the online controller. We must then use the k-memory controller or the memoryless controller.

7 Experimental results using Smacs

We have implemented the (non-blocking) memoryless algorithms in our tool SMACS (Symbolic MAsked Controller Synthesis), written in Objective CAML (OCaml (2005)), which uses the APRON library (Jeannet and Miné (2009)) and a generic fixpoint solver (FixPoint (2009)). It can be downloaded and used online from the SMACS web page (SMACS (2010)). Next, we define several examples and present the solutions that SMACS has computed for these systems in order to evaluate experimentally our method

7.1 Description of SMACS

We first illustrate the syntax of our tool SMACS by giving the source code of the producer and consumer example given in Fig.1.

Variables and control structure. Unlike the model of Def. 1, SMACS considers STS with explicit locations. There are two types of variables: (unbounded) integer or real (float). 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.

First, we define the events, specifying whether they are controllable (C) or uncontrollable (U). Then, we define the variables with their type, and the initial location (*Choice*).

```
events:
/* first producer/consumer */
C Choice_X;
U Cons;
C Stop_cons;
C Prod;
C Stop_prod;
/* second producer/consumer  */
C Choice_X_prime;
U Cons_prime;
C Stop_cons_prime;
C Prod_prime;
C Stop_prod_prime;
```

```
/* first producer */
int x ;
int y ;
/* second producer */
int x_prime ;
int y_prime ;

initial : Choice
```

Table 1: Declaration of the events and variables

Guards and assignments. The assignments are given by linear expressions; the guards are boolean combinations of linear constraints. The APRON library implements several numerical abstract lattices as intervals (Cousot and Cousot (1977)), octagons (Miné (2001)) and convex polyhedra (Cousot and Halbwachs (1978)). Those abstract lattices work well when the guards are linear constraints and the assignments are also linear. In each location, we give the list of transitions enabled in this location. Each transition has a destination ('to ...'), a guard ('when ...'), an event, and an optional assignment ('with ...'). A part of the source code describing the transitions of the producer and consumer example (see Fig. 1) is the following:

Bad states. In each location, the user can define a combination of linear constraints specifying the bad states. The source code describing the bad states of the producer and consumer example (see Example 3) is the following:

Masks. The user can define four kinds of masks:

1. *Indistinguishable locations:* the controller cannot distinguish some specified locations of the system.
2. *Hidden variables:* the controller has no information regarding the value of these variables.
3. *Partially hidden variables:* the value of a numerical variable is unknown if this value belongs to a specified interval (the user can specify an interval for each variable).

```
state Choice :
to PX : when true, Choice_X;
to PX_prim : when true, Choice_X_prime;

state PX:
to PX : when true, Prod with x = x+1;
to CX : when true, Stop_prod with y = 0;

state PX_prime:
to PX_prime : when true, Prod_prime with x_prime = x_prime+1;
to CX_prime : when true, Stop_prod_prime with y_prime = 2;
```

Table 2: Declaration of transition function

```
bad_states :
state CX : x<=10 and y <= 2
state CX_prime : x_prime <= 10 and y_prime <= 2
```

Table 3: Declaration of the Forbidden states

4. *Indistinguishable Intervals:* when the system is in a state $\boldsymbol{\nu} = \langle \boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_n \rangle$, the controller gets for each variable v_i a value in the interval $[\boldsymbol{\nu}_i - c_i, \boldsymbol{\nu}_i + c'_i]$, where c_i and c'_i are constants. Note that for $i \neq j$, the values c_i and c'_i can be different from c_j and c'_j.

The first three masks are partition of the state space and the last one is a covering. Masks are optional. If there is no mask specified, then the analysis is performed on a system under full observation.

```
controller 1 mask: intervals: (x, [5.0, 15.0])
```

Table 4: Declaration of the Mask

Output. The result of SMACS is a description of the function $\mathcal{F}$. It consists in displaying, for each action σ, the set of observation states, for which σ is forbidden. When the mask M is a partition of the state space, the controlled system can be computed and SMACS displays it graphically (when the mask is a covering, the controlled system cannot be finitely represented).

Remark 3 *Note that from a computational point of view, SMACS allows the use of three abstract lattices (depending of the precision the user want to obtain when computing the controller): the lattice of intervals, the lattice of octagons and the lattice of convex polyhedra (the lattice of octagons can be seen as a particular case of the lattice of convex polyhedra where the coefficients of the linear constraints are equal to* 1*).* $\diamond$

7.2 Experiments

Let us now report on our experiments.

Toy example. This example illustrates the algorithms given in sections 4 and 5. The STS has explicit locations $\ell \in \{\ell_i \mid i \in [0, 8]\}$ and two natural variables x and y (see Fig. 8). A system state is a triple $\langle \ell, x, y \rangle$. The initial condition is given by the state $\langle \ell_0, 0, 0 \rangle$. The set Bad is defined by $\{\langle \ell_6, k_1, k_2 \rangle | k_1, k_2 \in \mathbb{N}\}$. This example features:

- a loop of uncontrollable events, which implies that a naive exact computation of the fixpoint would requires 1000 computation steps, compared to only 2 computation steps if we apply a widening technique
- a controllable event that must be disabled under partial observation (for the mask considered), but which is enabled under full observation
- controllable events that must be disabled to prevent from reaching a deadlock state, when the deadlock free property must be ensured.

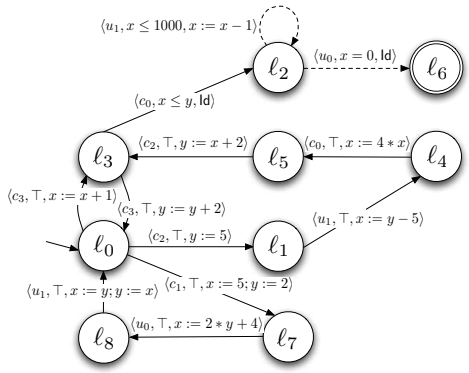

Fig. 8: Toy example

We consider several cases:

- In the easiest case (we only consider the basic control problem under full observation), the computation of $I(Bad)$ first detects that x must not be equal to 0 in location ℓ_2 because of the uncontrollable event u_0. This computation terminates with the set $I(Bad)$ given by: $0 \leq x \leq 1000$ in location ℓ_2 and every value of x and y in location ℓ_6. The controller thus disables the transition between locations ℓ_3 and ℓ_2 when $0 \leq x \leq 1000$.
- If the controller must ensure that the system is deadlock free, location ℓ_2 becomes *bad* without any condition on x. The controller thus totally disables the event c_0 in location ℓ_3.
- We now assume that the locations ℓ_3 and ℓ_4 return the same observations. When the controller must ensure that the system is deadlock free, it also disables the event c_0 in location ℓ_4, and thus the whole sequence $\ell_0 \xrightarrow{c_2} \ell_1 \xrightarrow{u_1} \ell_4 \xrightarrow{c_0} \ell_5 \xrightarrow{c_2} \ell_3$.

Cat and Mouse. The STS of Fig. 9 illustrates a modified version of the cat and mouse example given by Ramadge and Wonham (1987). The STS has explicit locations $\ell \in \{\mathsf{Awake}, \mathsf{Cheese}, \mathsf{Dead}, \mathsf{Sleep}, \mathsf{Tired}\}$ and two natural 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 $\langle \mathsf{Sleep}, 1, 0\rangle$. 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 Cheese, 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. The controllable (resp. uncontrollable) transitions are those drawn in plain (resp. dashed) lines. The set Bad is defined by $\{\langle \mathsf{Dead}, k_1, k_2\rangle | k_1, k_2 \in \mathbb{N}\}$ and thus $\mathrm{Coreach}_{uc}(Bad) = \{\langle \mathsf{Cheese}, k_1, k_2\rangle | (k_1 \in [0, 1000]) \wedge (k_2 \in \mathbb{N})\} \cup \{\langle \mathsf{Awake}, k_1, k_1\rangle | k_1 \in \mathbb{N}\} \cup \{\langle \mathsf{Dead}, k_1, k_2\rangle | k_1, k_2 \in \mathbb{N}\}$.

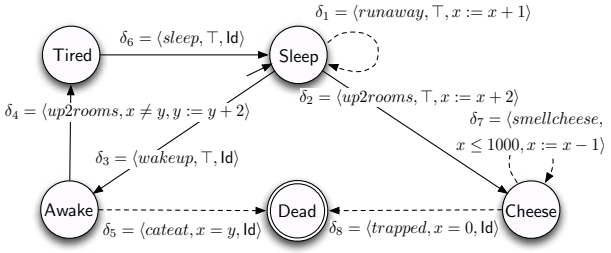

Fig. 9: The cat and mouse example

We consider three masks:

1. the first mask is the identity function (i.e. the system is under full observation). The controller prevents the cat to wake up (event *wakeup*) when she is in the same room than the mouse, and it also disables the event *up2rooms* in the location Sleep when $x \leq 998$.

2. the second mask is a covering $M : Loc \times \mathbb{N} \times \mathbb{N} \mapsto Loc \times \mathbb{N} \times \mathbb{N}$, where for each state $\boldsymbol{\nu} = \langle \ell, x, y \rangle \in \mathcal{D}_V$ the value of all possible observations of $\boldsymbol{\nu}$ the controller can have is given by $M(\langle \ell, x, y \rangle) = \{\langle \ell, x', y' \rangle | (x' \in [x-4, x+4]) \wedge (y' \in [y-4, y+4])\}$. The controller prevents the cat from awaking (event *wakeup*) when $0 \leq |x - y| \leq 8$, and it also disables the event *up2rooms* in the location Sleep when $x \leq 1002$.
3. for the third mask, y is an hidden variable and we ensure the deadlock free property. The controller disables the event *up2rooms* in the location Sleep without any condition and it prevents the cat from awaking.

Producer and consumer. This system was already defined in Example 3 and we consider three masks:

1. Under full observation, the controller forbids the action *Stop_prod* (resp. *Stop_prod'*) in the location PX (resp. PX′) when $x < 12$ (resp. $x' < 12$).
2. For the mask of Example 2, SMACS gives the same result than the one presented in Example 3.
3. The third mask $M : Loc \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mapsto Loc \times \mathbb{N} \times \mathbb{N}$ is defined, for all $\langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, by $M(\langle \ell, x, x', y, y' \rangle) = \langle \ell, x, y \rangle$, i.e. the controller does not observe the variables x' and y'. The controller forbids the action *Stop_prod* (resp. *Stop_prod'*) in the location PX (resp. PX′) when $x < 12$ (resp. $x' \in \mathbb{N}$).

Note that SMACS obtained all results in a few milliseconds on a Intel Core 2 Duo with 1 GB RAM.

Performance analysis w.r.t. the abstraction. Next, we consider more complex systems to be controlled by increasing the number of variables in the cat and mouse, toy and producer and consumer examples:

– Toy (Table 5): the number of variables of the system is increased and they are used in the uncontrollable transitions that can lead to forbidden states. The controller does not observe some of these new variables. With the lattices of intervals, SMACS uses few memory and computes the controllers fast. However, these controllers are very restrictive. With the lattices of convex polyhedra and octagons, SMACS computes maximal controllers and we can remark that the performances (time and memory) are better with the first lattice. For example, for the case with 220 variables, SMACS computes the controller in 36.1 seconds and uses 108 MB with the lattice of convex polyhedra, whereas it needs 601 seconds and 468 MB with the lattice of octagons to compute the controller. It is due to the fact that SMACS uses more linear constraints to represent a (abstract) set of states with the lattice of octagons.

Number of variables in the toy example	Convex polyhedra		Octagons		Intervals	
	Time	Memory (MB)	Time	Memory (MB)	Time	Memory (MB)
30	0.28	< 12	2.4	24	0.06	< 12
80	2.47	60	36.9	240	0.08	< 12
130	8.9	72	157.7	312	0.15	< 12
180	21.8	96	391.1	396	0.25	< 12
220	38.6	108	722	492	0.37	< 12

Table 5: Time (in seconds) and memory (in MB) used by SMACS to synthesize a controller for a modified version of the toy example.

– Cat and Mouse (Table 6): the number of variables of the system is increased by adding mice. The controller has a perfect observation of the system and must prevent the dead of all the mice. The better performances (time and memory) are obtained with the lattice of intervals, but the controllers that SMACS computes with this lattice are quite restrictive: the transition labelled by action *wakeup* is always forbidden by the controller with the lattice of intervals, while this transition is forbidden only when necessary with the lattice of convex polyhedra. Indeed, with the lattice of intervals, there is no good way to abstract a linear constraint on 2 or more variables (e.g. a guard $x \leq y$ or $x = y$). With the lattice of octagons, SMACS computes better controllers, but they are not maximal. Moreover, the performances (time and memory) are not satisfactory. SMACS only computes maximal controllers with the lattice of convex polyhedra and the performances obtained with this lattice are better than the ones obtained with the lattice of octagons.

Number of variables in the cat & Mouse example	Convex polyhedra		Octagons		Intervals	
	Time	Memory (MB)	Time	Memory (MB)	Time	Memory (MB)
10	0.08	< 12	0.3	< 12	0.03	< 12
40	0.8	36	5.8	216	0.06	< 12
90	4.3	60	58.6	228	0.15	< 12
140	14.3	84	210.3	372	0.22	< 12
200	36.1	108	601	468	0.41	< 12

Table 6: Time (in seconds) and memory (in MB) used by SMACS to synthesize a controller for a modified version of the cat and mouse example.

– Producer and Consumer: the system is made more complex by producing n (where $n > 0$) kinds of pieces $(X_i)_{i \leq n}$. The production of each kind of pieces requires the definition of two variables and the control requirements consist in ensuring that, for each kind of pieces, there are at least 11 pieces of this kind. Moreover, the controller does not observe the value of the variable X_0 when its value belongs to a certain interval. Again, the better performances (time and memory) are obtained with the lattice of intervals, but the controllers that SMACS obtains with this lattice are very rough. With the lattices of convex polyhedra and octagons, SMACS computes maximal controllers and we can remark that the performances (time and memory) are better with the first lattice. For example, for the case with 160 variables, SMACS computes the controller in 92.3 seconds and uses 624 MB with the lattice of convex polyhedra, whereas it needs 1181 seconds and 2124 MB with the lattice of octagons to compute the controller.

Number of variables in the producer and consumer example	Convex polyhedra		Octagons		Intervals	
	Time	Memory	Time	Memory	Time	Memory
40	1.26	96	7.8	408	0.15	< 12
80	8.8	168	82.4	876	0.61	< 12
120	33.4	288	381.6	1284	1.7	12
160	92.3	624	1181	2124	3.6	24
180	141.2	852	> 1200	–	5	< 36

Table 7: Time (in seconds) and memory (in MB) used by SMACS to synthesize a controller for a modified version of the producer and consumer example.

8 Conclusion and Future Works

In this paper, we propose algorithms for the synthesis of controllers through partial observation of infinite state systems modelled by STS. Those algorithms are *symbolic* and deal with the cases of memoryless controllers, controllers with memory and online controllers. One can notice that our algorithms can be used to enforce any *safety property*, because a safety control problem can be reduced to a state avoidance control problem. Our tool SMACS implements the algorithms to synthesize memoryless controllers and allowed us to make an empirical validation of our method and shows its feasibility and usability. To our knowledge, it is the first tool to synthesis an effective controller for STS. To overcome undecidability issue, our algorithms use abstract interpretation techniques that provide an overapproximation of the set $I(Bad)$ (or of the set $I_{bl}(Bad)$ for the deadlock free problem).

Further works will look at possible refinements in the abstract domain to obtain, when needed, more permissive controllers and we also want to implement other kinds of masks in SMACS. Moreover, we want to extend our method (and thus our tool) to the control of distributed systems. We also want to define and implement an automatic test generator using our framework.

References

Balemi S, Hoffmann G, Wong-Toi H, Franklin G (1993) Supervisory control of a rapid thermal multiprocessor. IEEE Transactions on Automatic Control 38(7):1040–1059

Bourdoncle F (1992) Sémantiques des langages impératifs d'ordre supérieur et interprétation abstraite. PhD thesis, Ecole Polytechnique

Brandt RD, Garg VK, Kumar R, Lin F, Marcus SI, Wonham WM (1990) Formulas for calculating supremal and normal sublanguages. Systems and Control Letters 15(8):111–117

Bryant R (1986) Graph-based algorithms for boolean function manipulations. IEEE Transaction on Computers C-45(8):677–691

Cassandras C, Lafortune S (2008) Introduction to Discrete Event Systems (2nd edition). Springer

Chatterjee K, Doyen L, Henzinger TA, Raskin JF (2007) Algorithms for omega-regular games of incomplete information. Logical Methods in Computer Science 3(3:4)

Cousot P, Cousot R (1977) Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL'77, pp 238–252

Cousot P, Halbwachs N (1978) Automatic discovery of linear restraints among variables of a program. In: POPL '78, pp 84–96, DOI http://doi.acm.org/10.1145/512760.512770

De Wulf M, Doyen L, Raskin JF (2006a) A lattice theory for solving games of imperfect information. In: Hespanha and Tiwari (2006), pp 153–168

De Wulf M, Doyen L, Raskin JF (2006b) A lattice theory for solving games of imperfect information. In: Hespanha and Tiwari (2006), pp 153–168

FixPoint (2009) Fixpoint: an OCaml library implementing a generic fix-point engine. Http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint/

Halbwachs N, Proy Y, Roumanoff P (1997) Verification of real-time systems using linear relation analysis. Formal Methods in System Design 11(2):157–185

Henzinger T, Majumdar R, Raskin JF (2005) A classification of symbolic transition systems. ACM Trans Comput Logic 6(1):1–32, DOI http://doi.acm.org/10.1145/1042038.1042039

Hespanha J, Tiwari A (eds) (2006) Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings, Lecture Notes in Computer Science, vol 3927, Springer

Hill R, Tilbury D, Lafortune S (2008) Covering-based supervisory control of partially observed discrete event systems for state avoidance. In: 9th International Workshop on Discrete Event Systems, pp 2–8

Jeannet B (2003) Dynamic partitioning in linear relation analysis. Application to the verification of reactive systems. Formal Methods in System Design 23(1):5–37

Jeannet B, Miné A (2009) Apron: A library of numerical abstract domains for static analysis. In: Bouajjani A, Maler O (eds) CAV, Springer, Lecture Notes in Computer Science, vol 5643, pp 661–667

Jeannet B, Jéron T, Rusu V, Zinovieva E (2005) Symbolic test selection based on approximate analysis. In: TACAS'05, Volume 3440 of LNCS, Edinburgh (Scottland), pp 349–364

Kalyon G, T LG, Marchand H, Massart T (2009) Control of infinite symbolic transition systems under partial observation. In: European Control Conference, Budapest, Hungary, pp 1456–1462

Kumar R, Garg V (2005) On computation of state avoidance control for infinite state systems in assignment program model. IEEE Trans on Automation Science and Engineering 2(2):87–91

Kumar R, Garg V, Marcus S (1993) Predicates and predicate transformers for supervisory control of discrete event dynamical systems. IEEE Trans Autom Control 38(2):232–247, URL `citeseer.ist.psu.edu/kumar95predicates.html`

Kupferman O, Madhusudan P, Thiagarajan P, Vardi M (2000) Open systems in reactive environments: Control and synthesis. In: Proc. 11th Int. Conf. on Concurrency Theory, Springer-Verlag, Lecture Notes in Computer Science, vol 1877, pp 92–107

Le Gall T, Jeannet B, Marchand H (2005) Supervisory control of infinite symbolic systems using abstract interpretation. In: Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05, pp 30–35

Lin F, Wonham W (1988) On observability of discrete-event systems. Inf Sci 44(3):173–198

Marchand H, Bournai P, Le Borgne M, Le Guernic P (2000) Synthesis of discrete-event controllers based on the signal environment. Discrete Event Dynamic System : Theory and Applications 10(4):347–368

Miné A (2001) The octagon abstract domain. In: Proc. of the Workshop on Analysis, Slicing, and Transformation (AST'01), IEEE CS Press, Stuttgart, Gernamy, IEEE, pp 310–319

Miremadi S, Akesson K, Fabian M, Vahidi A, Lennartson B (2008a) Solving two supervisory control benchmark problems using supremica. In: 9th International Workshop on Discrete Event Systems, pp 131–136

Miremadi S, Akesson K, Lennartson B (2008b) Extraction and representation of a supervisor using guards in extended finite automata. In: 9th International Workshop on Discrete Event Systems, pp 193–199

OCaml (2005) The programming language Objective CAML. Http://caml.inria.fr/

Pnueli A, Rosner R (1989) On the synthesis of an asynchronous reactive module. In: Ausiello G, Dezani-Ciancaglini M, Rocca SD (eds) ICALP, Springer, Lecture Notes in Computer Science, vol 372, pp 652–671

Ramadge P, Wonham W (1987) Modular feedback logic for discrete event systems. SIAM J Control Optim 25(5):1202–1218

Ramadge P, Wonham W (1989) The control of discrete event systems. Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems 77(1):81–98

Reif J (1984a) The complexity of two-player games of incomplete information. J Comput Syst Sci 29(2):274–301

Reif J (1984b) The complexity of two-player games of incomplete information. J Comput Syst Sci 29(2):274–301

SMACS (2010) The SMACS tool. `http://www.smacs.be/`

Takai S, Kodama S (1998) Characterization of all M-controllable subpredicates of a given predicate. International Journal of Control 70:541–549(9)

Takai S, Ushio T (2003) Effective computation of an $L_m(G)$-closed, controllable, and observable sublanguage arising in supervisory control. Systems and Control Letters 49(3):191–200

Tarski A (1955) A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5:285–309

Thistle J, Lamouchi H (2009) Effective control synthesis for partially observed discrete-event systems. SIAM J Control Optim 48(3):1858–1887

Wonham W, Ramadge P (1988) Modular supervisory control of discret-event systems. Mathematics of Control, Signals, and Systems 1(1):13–30

Yoo T, Lafortune S (2006) Solvability of centralized supervisory control under partial observation. Discrete Event Dyn Syst 16:527–553