Lazy Controller Synthesis for Monotone Transition Systems and Directed Safety Specifications

Elena A. Ivanova, Adnane Saoud, Antoine Girard

Lazy Controller Synthesis for Monotone Transition Systems and Directed Safety Specifications ⋆

Elena Ivanova[a,⋆⋆], Adnane Saoud[b,⋆⋆], and Antoine Girard[a]

[a] *Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 3, rue Joliot-Curie, 91190, Gif-sur-Yvette, France.*

[b] *Department of Electrical Engineering & Computer Sciences, University of California, Berkeley, 569 Cory Hall, Berkeley, CA, 94720, USA*

Abstract

In this paper, we provide a lazy control synthesis algorithm for monotone transition systems and directed safety specifications. Two classes of monotone transition systems are presented: state monotone transition systems and input-state monotone transition systems. For the first class of systems, a partial order is defined only on the state space. For the second, the input space is ordered as well. The introduced lazy synthesis approach is based on the efficient computation of predecessors. It benefits not only from a monotone property of transition systems but also from the ordered structure of the state (input) space and the fact that directed safety specifications are considered. To enrich the class of the considered specifications, we also present an incremental controller synthesis framework, which allows us to deal with intersections of upper and lower-closed safety requirements. We then compare the proposed approach with the classical safety synthesis algorithm and illustrate the advantages, in terms of run-time and memory efficiency, on an adaptive cruise control problem.

Key words: Monotone transition systems; monotone dynamical systems; directed safety specifications; lazy controller synthesis; symbolic control.

1 Introduction

Abstraction-based synthesis techniques have been an ongoing research area in the last decade (see e.g., [1,2] and the references therein). They consist in creating a finite-state abstraction (or a symbolic model) for a continuous or a hybrid system and refining the controller synthesized for the abstraction to a controller for the original system. The replacement of a dynamical system by its abstraction principally enables the use of various techniques developed in the area of supervisory control of discrete event systems [3].

Symbolic models are often obtained through discretization of the state and input spaces. Consequently, most of the abstraction-based approaches do not scale well (see e.g. [4,5] and the references therein).

To tackle the scalability problem different ideas have been proposed. In [6,7] optimal abstraction parameters are derived to minimize the size of symbolic models. In [8–10], compositional approaches are used to improve the scalability of symbolic control techniques. In [11–14], the authors proposed multi-scale abstractions, where sampling parameters can be refined during the synthesis process. The general idea for these papers is to start with a coarse abstraction and then iteratively refine it. While results in [11,12] are restricted for linear and incrementally stable systems, respectively, the approaches provided in [13,14] are applicable for general classes of nonlinear systems. The authors in [12–17] synthesize controllers lazily [1] . In [12–14] the authors proposed to re-

⋆ A preliminary version of this paper was presented at the 58th IEEE Conference on Decision and Control. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725144). Corresponding author : A. Saoud ; e-mail: Elena.Ivanova@l2s.centralesupelec.fr, asaoud@berkeley.edu, Antoine.Girard@l2s.centralesupelec.fr

⋆⋆The authors contributed equally to this work.

[1] In classical approaches [1], one first constructs the whole abstraction for the original system and then use the precomputed abstraction to synthesize the controller. In lazy approaches, the system is abstracted in parallel to the controller synthesis procedure. Moreover, we only compute the

fine the abstraction locally. Moreover, in [12,14] the abstraction is computed for states that are reachable from the initial set. Proposed for uniform abstractions the approaches from [15–17] can be combined with multi-scale ideas as well. In [15], the authors present a lazy synthesis algorithm for safety and reachability using three-valued abstractions. In [16], a lazy safety controller for event-based symbolic models of incrementally stable switched systems is provided. In [17], adaptive time sampling approach allow the authors to iteratively explore only boundary of the controllable domain.

In this paper, we present a (lazy) synthesis algorithm for monotone transition systems and directed safety specifications. The class of monotone transition systems is of practical interest since it arises from monotone dynamical systems, which frequently appear in engineering applications such as traffic networks [18], biological networks [19] and power systems [20]. We consider two different classes of systems: state monotone transition systems and input-state monotone transition systems. For the former only the state space is equipped with a partial order. For the latter, both state and input spaces are equipped with a partial order. First, we focus on directed safety specifications and provide an approach based on the concept of a basis to compute the set of predecessors lazily. Then we present an incremental synthesis procedure allowing us to deal with the intersection of upper and lower-closed safety specifications. Finally, we consider an adaptive cruise control problem. We show that, while ensuring completeness with respect to the classical safety synthesis algorithm, the lazy approach allows us to speed up the computations and reduce memory consumption.

In spirit, the closest works in the literature are [21,22]. In [21], sparse abstractions were proposed for monotone dynamical systems and directed specifications. We go one step further by providing a lazy synthesis algorithm benefiting from a particular structure of the considered problem. In [22], the authors introduce a notion of s-sequence to characterize a controlled invariant of the system. This notion is relatively close to the notion of basis, which we use to describe a predecessor operator in our work. However, the results presented in [22] are restricted by the class of cooperative system, while we address a more general class of monotone systems. Moreover, looking at the computation of controlled invariants, as an optimization problem, [22] do not guarantee the maximality of the obtained control invariant set. They also propose to use a simple open-loop control policy to keep a trajectory within a safe set. Although such an approach is memory efficient, it hardly possible to use their controller as a start point for more general tasks (for example, obstacle-avoided reachability specification or optimal control problems with hard constraints). In our paper, we first use a fixed-point approach to compute the maximal controlled invariant and then construct the maximal safety controller, which brings us to a more general solution.

fragment of the abstraction that is essential for the controller synthesis.

A preliminary version of this work has been presented in the conference paper [23]. The current paper extends the approach in different directions: First, while in [23], we have only provided the results for input-state monotone transition systems, in this paper, we also deal with the more general class of state monotone transition systems. Second, in [23], we have considered only directed specifications. However, in this paper, we also explain how to deal with safety specifications given by an intersection of upper and lower-closed sets. Third, the proofs of different results are simplified, since now they are based on the predecessor operator notion. Finally, the numerical example is improved, and we are evaluating the performance of our approach not only using a runtime comparison but also in terms of memory efficiency, which has not been done in the conference version.

The paper is organized as follows. In Section 2, some required preliminaries are provided. In Section 3, we introduce different classes of monotone transition systems. In Section 4, we present a lazy algorithm to compute the set of predecessors. In Section 5, we present a lazy synthesis algorithm for monotone transition systems and lower-closed safety specifications. In Section 6, we present an incremental approach to synthesize controllers for intersections of upper and lower-closed safety specifications. In Section 7, we show how the monotonicity property is preserved when going from dynamical systems to their symbolic abstractions. Finally, in Section 8, an illustrative example is proposed in order to show the efficiency of the proposed approach.

2 Preliminaries

2.1 Partial orders

A binary relation $\leq_{\mathcal{L}} \subseteq \mathcal{L} \times \mathcal{L}$ is a partial order if and only if for all $l_1, l_2, l_3 \in \mathcal{L}$ we have: (i) $l_1 \leq_{\mathcal{L}} l_1$, (ii) if $l_1 \leq_{\mathcal{L}} l_2$ and $l_2 \leq_{\mathcal{L}} l_1$ then $l_1 =_{\mathcal{L}} l_2$ and, (iii) if $l_1 \leq_{\mathcal{L}} l_2$ and $l_2 \leq_{\mathcal{L}} l_3$ then $l_1 \leq_{\mathcal{L}} l_3$. If neither $l_1 \leq_{\mathcal{L}} l_2$ nor $l_2 \leq_{\mathcal{L}} l_1$ holds, we say that l_1 and l_2 are incomparable. The set of all incomparable couples in $\mathcal{L}$ is denoted by $\mathrm{Inc}_{\mathcal{L}}$. We say that $l_1 <_{\mathcal{L}} l_2$ iff $l_1 \leq_{\mathcal{L}} l_2$ and $l_1 \neq_{\mathcal{L}} l_2$. We define $\geq_{\mathcal{L}}$ so that $l_1 \geq_{\mathcal{L}} l_2$ if and only if $l_2 \leq_{\mathcal{L}} l_1$.

For a partially ordered set $\mathcal{L}$, closed intervals are $[x, y]_{\mathcal{L}} = \{z \mid x \leq_{\mathcal{L}} z \leq_{\mathcal{L}} y\}$. Given a partially ordered set $\mathcal{L}$, for $a \in \mathcal{L}$ let $\downarrow a = \{x \in \mathcal{L} \mid x \leq_{\mathcal{L}} a\}$ and $\uparrow a = \{x \in \mathcal{L} \mid a \leq_{\mathcal{L}} x\}$. When $A \subseteq \mathcal{L}$ then its lower closure (respectively upper closure) is $\downarrow A = \bigcup_{a \in A} \downarrow a$ (respectively $\uparrow A = \bigcup_{a \in A} \uparrow a$). A subset $A \subseteq \mathcal{L}$ is said to be lower-closed (respectively upper-closed) if $\downarrow A = A$ (respectively $\uparrow A = A$).

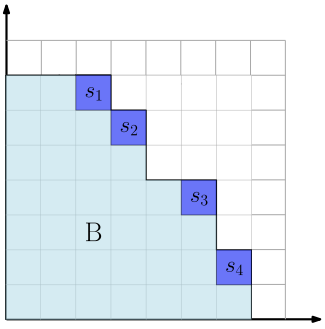


Fig. 1. Illustration of Definition 2. A lower-closed set B and its basis $\mathrm{Bas}(B) = \{s_1, s_2, s_3, s_4\}$. The state-space is equipped with the component-wise partial order $\leq$ defined on $\mathbb{R}^2$ and we have: $B = {\downarrow} B = {\downarrow} \mathrm{Bas}(B)$.

Definition 1 *Let $\mathcal{L}$ be a partially ordered set and $A \subseteq \mathcal{L}$ is a finite subset. The set of minimal elements of A is defined as* $\min(A) = \{x \in A \mid \forall x_1 \in A,\ x \leq_{\mathcal{L}} x_1 \text{ or } (x, x_1) \in Inc_{\mathcal{L}}\}$. *Similarly, the set of maximal elements of A is defined as* $\max(A) = \{x \in A \mid \forall x_1 \in A,\ x \geq_{\mathcal{L}} x_1 \text{ or } (x, x_1) \in Inc_{\mathcal{L}}\}$.

Most of the paper's statements are formulated for finite sets since we do not need the more general theory for our purposes.

Proposition 1 ([24]) *Let $\mathcal{L}$ be a partially ordered set, $A_i \subseteq \mathcal{L}$ are finite subsets satisfying $A_i = {\downarrow} A_i$ for all $i \in \{1, \ldots, p\}, p \in \mathbb{N}_{\geq 1}$. Then ${\downarrow}(\cup_{i=1}^{p} A_i) = \cup_{i=1}^{p} A_i$ and ${\downarrow}(\cap_{i=1}^{p} A_i) = \cap_{i=1}^{p} A_i$.*

Intuitively, the result of Proposition 1 means that a union (an intersection) of lower-closed sets is lower-closed.

For finite lower-closed sets, we use the operator max to introduce the notion of the basis [25], which serves as a simpler representation of the lower-closed set.

Definition 2 *Let $\mathcal{L}$ be a finite partially ordered set. Let $Z \subseteq \mathcal{L}$ be a lower-closed set. A set $B = \{s_1, \ldots, s_N\} \subseteq Z$ is said to be the basis of Z, denoted $B = Bas(Z)$, if $B = \max(Z)$ or in other words*

- $Z = \bigcup_{i=1,\ldots,N} {\downarrow} s_i$;
- *for all $s_i, s_j \in B$, if $s_i \neq s_j$ then $(s_i, s_j) \in Inc_{\mathcal{L}}$.*

The existence and uniqueness of a finite basis of a finite lower-closed set follow from the fact that the relation $\leq_{\mathcal{L}}$ is a well-quasi-order [26]. An illustration of the concept of basis is given in Figure 1.

2.2 Transition systems

Definition 3 *A transition system is a tuple $T = (X, U, \Delta)$, where X is a set of states, U is a set of inputs and $\Delta \subseteq X \times U \times X$ is a transition relation.*

A transition system is said to be finite if X and U are finite. A transition system is said to be deterministic if $\mathrm{card}(\Delta(x, u)) \leq 1$ for all $x \in X$ and $u \in U$. We introduce notation $x' \in \Delta(x, u)$ as an alternative representation for a transition $(x, u, x') \in \Delta$ and we call the state x' u-successor of the state x, while x is u-predecessor of state x' correspondingly. For $A \subseteq X$ and $V \subseteq U$, we denote by $\Delta(A, V) = \bigcup_{a \in A} \bigcup_{v \in V} \Delta(a, v)$. For the transition system S, we assume that for all $x \in X$ and for all $u \in U$, $\Delta(x, u) \neq \emptyset$. This means that for any state all the inputs are enabled.

3 Monotone transition systems

In this section, we introduce the class of monotone transition systems that preserve order on input and state spaces. We then provide necessary and sufficient conditions for a transition system to be monotone. We first start with the class of transition systems, where only the set of states is partially ordered.

Definition 4 *Consider a transition system $T = (X, U, \Delta)$ where the set of states X is equipped with a partial order $\leq_X$. The transition system T is said to be:*

- *Lower state monotone (LSM) if for all $x_1, x_2 \in X$, for all $u \in U$, with $x_1 \leq_X x_2$, it follows, that for any $x_1' \in \Delta(x_1, u)$, there is $x_2' \in \Delta(x_2, u)$, such that $x_1' \leq_X x_2'$;*
- *Upper state monotone (USM) if for all $x_1, x_2 \in X$, for all $u \in U$, with $x_1 \leq_X x_2$, it follows, that for any $x_2' \in \Delta(x_2, u)$, there is $x_1' \in \Delta(x_1, u)$, such that $x_1' \leq_X x_2'$.*

The transition system T is said to be state monotone (SM) if it is both LSM and USM.

One can readily see that the concepts of USM and LSM coincide when the transition system T is deterministic.

We then consider the class of transition systems where both state and input spaces are partially ordered.

Definition 5 *Consider a transition system $T = (X, U, \Delta)$ where the set of states X and the set of inputs U are equipped with partial orders $\leq_X$, $\leq_U$, respectively. The transition system T is said to be:*

- *Lower input-state monotone (LISM) if for all $x_1, x_2 \in X$, for all $u_1, u_2 \in U$, with $x_1 \leq_X x_2$ and $u_1 \leq_U u_2$, it follows that for all $x_1' \in \Delta(x_1, u_1)$, there is $x_2' \in \Delta(x_2, u_2)$ such that $x_1' \leq_X x_2'$;*
- *Upper input-state monotone (UISM) if for all $x_1, x_2 \in X$, for all $u_1, u_2 \in U$, with $x_1 \leq_X x_2$ and $u_1 \leq_U u_2$, it follows that for all $x_2' \in \Delta(x_2, u_2)$, there is $x_1' \in \Delta(x_1, u_1)$ such that $x_1' \leq_X x_2'$.*

The transition system T is said to be input-state monotone (ISM) if it is both LISM and UISM.

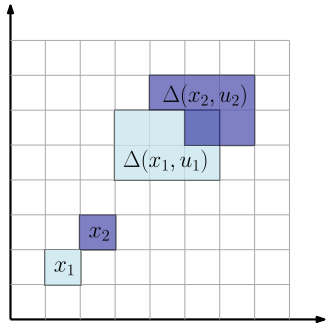


Fig. 2. Illustration of Theorem 1. Given two states $x_1 \leq_X x_2$ and two inputs $u_1 \leq_U u_2$, if the transition system is LISM then we have that $\Delta(x_1, u_1) \subseteq\downarrow \Delta(x_2, u_2)$.

Similarly to the case of SM transition systems, the concepts of UISM and LISM coincide when the transition system T is deterministic.

It follows straightforward from the definitions above that any ISM transition system is a SM transition system. Moreover, any SM system can be seen as ISM, with a partial order defined as $u \leq_U u' \Leftrightarrow u = u'$. However, this trivial order does not have any practical interest, and while speaking about ISM transition systems, we assume that there are at least two elements in $u, u' \in U$, such that $u <_U u'$. Let us provide an example, illustrating the difference between SM and ISM systems.

Example 1 *Let us consider the transition system* $T = (X, U, \Delta)$ *where:*

- *the sets of states* $X = \mathbb{R}^2$*;*
- *the set of inputs* $U = \{1, 2\}$*;*
- *the transition relation: for* $x \in X$ *and* $u \in U$

$$\Delta(x, u) = \begin{cases} A_1 x & \text{if } u = 1 \\ A_2 x & \text{if } u = 2 \end{cases}$$

where $A_1 = \begin{pmatrix} 0.1 & 0.9 \\ 3 & 0.7 \end{pmatrix}$ *and* $A_2 = \begin{pmatrix} 0.2 & 2 \\ 0.1 & 0.7 \end{pmatrix}$

We can remark that the transition system presented above is SM, while it is not ISM for any non-trivial partial order on U*.*

Until Section 7, we work only with lower-closed sets and lower (input-)state monotone transition systems while keeping in mind that analogous results can be formulated for upper-closed sets and upper (input-)state monotone transition systems as well.

Now, let us give some characterizations of LISM transition systems. We start with an auxiliary lemma.

Lemma 1 *Let* $\mathcal{L}$ *be a partially ordered set and* $A, B \subseteq \mathcal{L}$*. The set* A *is included in the lower closure of the set* B *(i.e.* $A \subseteq\downarrow B$*) if and only if for any* $a \in A$*, there exists* $b \in B$ *such that* $a \leq_X b$*.*

The proof follows immediately from the fact that for any set $B \subseteq \mathcal{L}$ we have $\downarrow B = \{x \in \mathcal{L} \mid \exists\, b \in B \text{ s.t. } x \leq_X b\}$.

Theorem 1 *For a transition system* $T = (X, U, \Delta)$ *the following statements are equivalent:*

(i) T *is a LISM transition system;*
(ii) for all $x_1, x_2 \in X$*, for all* $u_1, u_2 \in U$*, if* $x_1 \leq_X x_2$ *and* $u_1 \leq_U u_2$ *then* $\Delta(x_1, u_1) \subseteq\downarrow \Delta(x_2, u_2)$*;*
(iii) for all $x \in X$*, for all* $u \in U$ *we have:* $\Delta(\downarrow x, \downarrow u) \subseteq\downarrow \Delta(x, u)$*.*

PROOF. $(i) \Leftrightarrow (ii)$: Let $x_1, x_2 \in X$ and $u_1, u_2 \in U$ with $x_1 \leq_X x_2$ and $u_1 \leq_U u_2$. From Lemma 1, we have that $\Delta(x_1, u_1) \subseteq\downarrow \Delta(x_2, u_2)$ if and only if for any $x_1' \in \Delta(x_1, u_1)$, there exists $x_2' \in \Delta(x_2, u_2)$ with $x_1' \leq_X x_2'$. Hence, $(i) \Leftrightarrow (ii)$.

$(ii) \Rightarrow (iii)$: Let $x \in X$, $u \in U$, $x_1 \in (\downarrow x)$ and $u_1 \in (\downarrow u)$. We have $x_1 \leq_X x$ and $u_1 \leq_U u$. Hence, from (ii) we have that $\Delta(x_1, u_1) \subseteq\downarrow \Delta(x, u)$, for any $x_1 \in (\downarrow x)$ and any $u_1 \in (\downarrow u)$. Then, $\Delta(\downarrow x, \downarrow u) \subseteq\downarrow \Delta(x, u)$.

$(iii) \Rightarrow (ii)$: Let $x_1, x_2 \in X$ and $u_1, u_2 \in U$ with $x_1 \leq_X x_2$ and $u_1 \leq_U u_2$. We have that $x_1 \in (\downarrow x_2)$ and $u_1 \in (\downarrow u_2)$. Hence, from (iii), we have that $\Delta(x_1, u_1) \subseteq \Delta(\downarrow x_2, \downarrow u_2) \subseteq\downarrow \Delta(x_2, u_2)$. □

A graphical representation of the conditions in Theorem 1 is provided in Figure 2. We then have the following corollary for LSM transition systems.

Corollary 1 *For a system* $T = (X, U, \Delta)$ *the following statements are equivalent:*

(i) T *is a LSM transition system;*
(ii) for all $x_1, x_2 \in X$*, for all* $u \in U$*, if* $x_1 \leq_X x_2$ *then* $\Delta(x_1, u) \subseteq\downarrow \Delta(x_2, u)$*;*
(iii) for all $x \in X$*, for all* $u \in U$ *we have:* $\Delta(\downarrow x, u) \subseteq\downarrow \Delta(x, u)$*.*

4 Predecessors for lower-closed sets

In this section, we present a lazy approach to compute the predecessor operator for lower monotone transition systems.

4.1 Predecessor operator

Consider a transition system $T = (X, U, \Delta)$, let $A, B \subseteq X$ and $V \subseteq U$, we define the Pre operator as

$$\text{Pre}(A, V, B) = \{x \in A \mid \exists u \in V,\ \Delta(x, u) \subseteq B\} \quad (1)$$

Intuitively, $\mathrm{Pre}(A, V, B)$ contains all the states that are initially in $x \in A$ and for which all the successors can be steered in B by selecting inputs from V. The Pre operator is a fundamental tool for reachability analysis, which is commonly used for controller synthesis of transition systems [1]. Let us give some characterizations for the Pre operator for LSM transition systems and lower-closed sets.

Lemma 2 *Let $T = (X, U, \Delta)$ be a LSM transition system, consider the lower-closed sets $Z_1, Z_2 \subseteq X$ and let $V \subseteq U$. We have that $Pre(Z_1, V, Z_2)$ is a lower-closed set.*

PROOF. Let $x \in \mathrm{Pre}(Z_1, V, Z_2)$, then $x \in Z_1$ and there exists $u \in V$ such that $\Delta(x, u) \subseteq Z_2$. Let $x' \leq_X x$. Since Z_1 is a lower-closed set, then $x' \in Z_1$. Moreover, we have from (iii) in Corollary 1 that $\Delta(x', u) \subseteq\downarrow \Delta(x, u) \subseteq\downarrow Z_2 = Z_2$. The end of the proof follows straightforward from the definitions. □

Since the Pre operator has a lower-closed structure when dealing with lower-closed sets, we can use the notion of basis (see Definition 2) to describe the Pre operator for LSM transition systems.

Theorem 2 *Let $T = (X, U, \Delta)$ be a LSM transition system. Consider the lower-closed sets $A, Z_1, Z_2 \subseteq X$ such that $A \subseteq Z_1$ and the set $V \subseteq U$. We have $A \subseteq Pre(Z_1, V, Z_2)$ if and only if the following condition is satisfied:*

$$\forall x \in Bas(A),\ \exists u \in V\ s.t\ \Delta(x, u) \subseteq Z_2. \qquad (2)$$

PROOF. Let $A \subseteq Z_1$, we first assume that $A \subseteq \mathrm{Pre}(Z_1, V, Z_2)$. Since $\mathrm{Bas}(A) \subseteq A$, (2) is directly satisfied. Now let us prove the second implication. Assuming that (2) is satisfied, let us prove that $A \subseteq \mathrm{Pre}(Z_1, V, Z_2) = \{x \in Z_1 \mid \exists u \in U = V,\ \Delta(x, u) \subseteq Z_2\}$. First, we have that $A \subseteq Z_1$. For $x \in A$, there exists $x' \in \mathrm{Bas}(A)$ such that $x \leq_X x'$. Hence, we have from (2) the existence of $u \in V$ such that $\Delta(x', u) \subseteq Z_2$. Since $x \leq_X x'$, we have from (iii) in Corollary 1 that $\Delta(x, u) \subseteq\downarrow \Delta(x', u) \subseteq\downarrow Z_2 = Z_2$. Hence, $A \subseteq \mathrm{Pre}(Z_1, V, Z_2)$. □

Intuitively, the set $\mathrm{Pre}(Z_1, V, Z_2)$ can be seen as the maximal lower-closed subset of Z_1 for which condition (2) is satisfied.

Remark 1 *The result of Theorem 2 provides guidelines towards a lazy computation of the Pre operator. Indeed, while in the classical approach the condition $\Delta(x, v) \subseteq B$ is checked for all elements $x \in A$ (see equation (1)), it can be checked only for the elements $x \in Bas(A)$ when dealing with a lower-closed set of predecessors (see equation (2)).*

Algorithm 1 Pre(Z_1, V, Z_2)

Input: A LSM transition system $T = (X, U, \Delta)$, a lower-closed initial set $Z_1 \subseteq X$, a lower-closed final set $Z_2 \subseteq X$ and a subset of control inputs $V \subseteq U$.
Output: $\mathrm{Pre}(Z_1, V, Z_2)$.
1 **begin**
2 | $S^{unc} := \emptyset$; $B^c := \emptyset$;
3 | $B^{ex} = Bas(Z_1)$;
4 | **while** $B^{ex} \neq \emptyset$ **do**
5 | | $B^r := \{q \in B^{ex} \mid \exists u \in V \colon \Delta(q, u) \in Z_2\}$;
6 | | $B^c := B^c \cup B^r$;
7 | | $S^{unc} := S^{unc} \cup (B^{ex} \setminus B^r)$;
8 | | $B^{ex} := Bas(Z_1 \setminus S^{unc}) \setminus B^c$;
9 | **return** $\downarrow B^c$;

4.2 Lazy computation of predecessors

In this part, we propose a lazy fixed-point algorithm for the computation of a predecessor set. The algorithm is based on condition (2) and deals only with the elements of the basis in each iteration.

The inputs to Algorithm 1 are a LSM transition system $T = (X, U, \Delta)$, an initial lower-closed set $Z_1 \subseteq X$, a final lower-closed set $Z_2 \subseteq X$ and a subset of control inputs V, and the objective is to compute the basis of $\mathrm{Pre}(Z_1, V, Z_2)$, as a simple representation of $\mathrm{Pre}(Z_1, V, Z_2)$. Algorithm 1 works as follows: first we initialize the set B^{ex} with the basis of the set Z_1. Then at every iteration of the while loop 4-8, we explore all elements in B^{ex}. The controllable states, which satisfy (2), are accumulated in B^c. So, $B^r \subseteq \mathrm{Pre}(Z_1, V, Z_2)$ and, as a consequence, the set $B^c \subseteq \mathrm{Pre}(Z_1, V, Z_2)$ (see line 6). The uncontrollable states in Z_1, which do not belong to $\mathrm{Pre}(Z_1, V, Z_2)$, are stored in S^{unc} (line 7). The set B^c coincides with the basis of $\mathrm{Pre}(Z_1, V, Z_2)$, since at every iteration of the loop 4-8 we reinitialize the set B^{ex} (line 8) with a basis of elements in Z_1, which have not been marked as uncontrollable. However among these elements there are those, which have been already explored (they belongs to B^c), and we remove them to avoid the additional computations (see line 8). Let us also point out that in line 5 as soon as we find an acceptable input we do not explore the other inputs.

In the classical approach, to find $\mathrm{Pre}(Z_1, V, Z_2)$ we should check for every $x \in Z_1$ if there is $u \in V$ such that $\Delta(x, u) \subseteq Z_2$. However, in Algorithm 1, we leave all elements, which are smaller than the basis of $\mathrm{Pre}(Z_1, V, Z_2)$, unexplored. Indeed, we go through all states in Z_1 only if $\mathrm{Pre}(Z_1, V, Z_2)$ is empty. This laziness gives us an efficiency gain. An illustration of the execution of Algorithm 1 is provided in Figure 3.

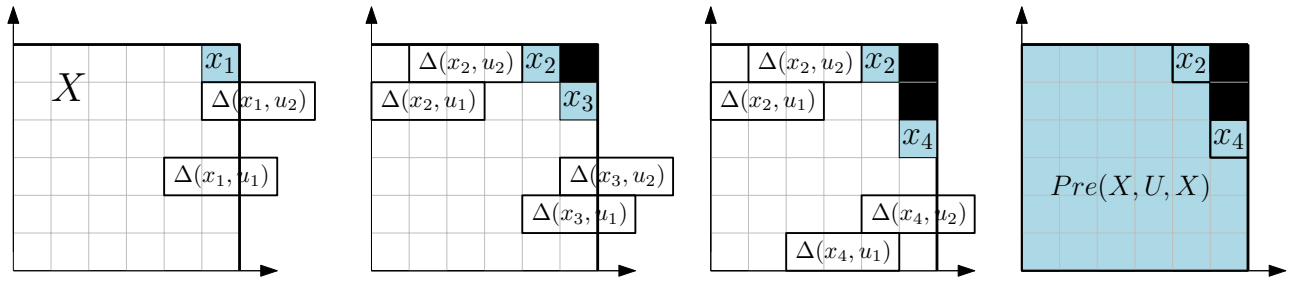


Fig. 3. Illustration of Algorithm 1. A monotone transition system $T = (X, U, \Delta)$ with $X = \downarrow X = \downarrow x_1$ and $U = \{u_1, u_2\}$, where the objective is to compute $\mathrm{Pre}(X, U, X)$. We start the exploration from the state $x_1 \in X$. Since x_1 does not satisfy condition (2), the state x_1 is then marked as uncontrollable (see line 7 in Algorithm 1) and represented in black in the figure. Then, we move to the next basis, which is made of the state x_2 and x_3. In the second iteration we explore the elements x_2 and x_3. After exploration, we mark the state x_2 as controllable and x_3 as uncontrollable. We then construct a new basis, made of two states x_2 and x_4. In the third iteration we only explore x_4 (x_2 has been explored in the second iteration). Since both x_2 and x_4 are controllable, Algorithm 1 terminates and provides an output given by $\mathrm{Pre}(X, U, X) = \downarrow x_2 \cup \downarrow x_4$ presented in the right figure with a blue color.

5 Controller synthesis for lower-closed safety specifications

Considering a transition system T and a safety specification $X^S \subseteq X$, we solve, in this section, a synthesis problem that consists in determining a controller, which keeps the trajectories of the system inside a safe set X^S.

Given a transition system $T = (X, U, \Delta)$, a controller for T is a set-valued map $C : X \rightrightarrows U$ and its domain is defined as $\mathrm{dom}(C) = \{x \in X \mid C(x) \neq \emptyset\}$. Particularly, a safety controller can be defined as follows:

Definition 6 *A safety controller C for the transition system $T = (X, U, \Delta)$ and the safe set X^S satisfies:*

- *$dom(C) \subseteq X^S$;*
- *$\forall x \in dom(C)$ and $\forall u \in C(x)$, $\Delta(x, u) \subseteq dom(C)$.*

There are, in general, several controllers that solve the safety problem. The maximal safety controller C^* is a safety controller such that for any other safety controller C and for all $x \in X$, we have $C(x) \subseteq C^*(x)$. The maximal safety controller is the best possible safety controller in the sense that any other controller solving the same safety problem would be more restrictive.

To find the maximal safety controller one can use the classical safety game [1], which consists of two steps. First, the domain of the maximal safety controller $\mathrm{dom}(C^*)$ is computed based on a fixed-point algorithm (see Algorithm 2). Then for every $x \in \mathrm{dom}(C^*)$ the maximal safety controller C^* is defined as follows:

$$C^*(x) = \{u \in U \mid \Delta(x, u) \subseteq \mathrm{dom}(C^*)\}. \tag{3}$$

In the next section we show how to speed up these two steps by benefiting from the monotone structure of the considered transition system.

Algorithm 2 : dom(C^*)

Input: A transition system $T = (X, U, \Delta)$ and a safe set $X^S \subseteq X$.
Output: $\mathrm{dom}(C^*)$.
1 **begin**
2 | $Z := \mathrm{Pre}(X^S, U, X^S)$;
3 | $Z^{pr} = X^S$;
4 | **while** $Z^{pr} \neq Z$ **do**
5 | | $Z^{pr} := Z$;
6 | | $Z := \mathrm{Pre}(Z, U, Z)$;
7 | **return** Z;

5.1 State monotone transition systems

5.1.1 Domain of the controller

First, let us characterize the domain of the maximal safety controller for LSM transition systems and lower-closed safety specifications $X^S \subseteq X$.

Proposition 2 *Let C^* be the maximal safety controller for a LSM transition system $T = (X, U, \Delta)$ and lower-closed safety specification $X^S \subseteq X$. Then the following properties hold:*

(i) $dom(C^*)$ is lower-closed w.r.t to the partial order $\leq_X$;
(ii) for all $x_1, x_2 \in X$, if $x_1 \leq_X x_2$ then $C^*(x_2) \subseteq C^*(x_1)$.

PROOF. (i) The proof follows immediately from Algorithm 2 and Lemma 2.

(ii) Let $x_1, x_2 \in X$ with $x_1 \leq_X x_2$. Let $u \in C^*(x_2)$. Then, $\Delta(x_2, u) \subseteq \mathrm{dom}(C^*)$. Hence, we have that $\Delta(x_1, u) \subseteq \downarrow \Delta(x_2, u) \subseteq \downarrow \mathrm{dom}(C^*) = \mathrm{dom}(C^*)$, where the first inclusion comes from the fact that T is a LSM transition system and the last equality comes from (i). Hence, by maximality of C^*, we have that $u \in C^*(x_1)$. Then, $C^*(x_2) \subseteq C^*(x_1)$. □

One can use the classical Algorithm 2 to find $\mathrm{dom}(C^*)$. However, in our case, Algorithm 2 is based on lazy computation of Pre operator, given by Algorithm 1. Let us remark that we can check if $Z^{pr} \neq Z$ by simply comparing the basis of the considered sets since they are lower-closed.

5.1.2 Maximal safety controller

Once the domain of the maximal safety controller is obtained, we rely on the following result to compute the maximal safety controller C^*:

Theorem 3 *Let C^* be the maximal safety controller for a LSM transition system $T = (X, U, \Delta)$ and lower-closed safety specification $X^S \subseteq X$. Let $U = \{u_1, \ldots, u_N\}$ be the set of inputs. Let the set $Z_i \subseteq X$ be defined as $Z_i = Pre(dom(C^*), u_i, dom(C^*))$. Then the following holds:*

$$u_i \in C^*(x) \Leftrightarrow x \in Z_i \quad \text{for } i \in \{1, \ldots, N\}$$

PROOF. Indeed $u_i \in C^*(x)$ is equivalent to $x \in \mathrm{dom}(C^*)$ and $\Delta(x, u_i) \in \mathrm{dom}(C^*)$, which is equivalent to $x \in \mathrm{Pre}(\mathrm{dom}(C^*), u_i, \mathrm{dom}(C^*)) = Z_i$. □

It can be seen that the computation of the maximal safety controller C^* is different and more efficient than the one used in the classical synthesis (see equation (3)). Indeed, in the classical case, one explores all the states in $\mathrm{dom}(C^*)$ and all the inputs $u_i \in U$. While in our approach, all inputs are explored, but not necessarily all the states. The main idea is that for each input $u_i \in U$, we only have to compute the set $Z_i = \mathrm{Pre}(\mathrm{dom}(C^*), u_i, \mathrm{dom}(C^*))$, and that can be done efficiently by using Algorithm 1.

5.2 Input-state monotone transition systems

In this part, we propose a lazy safety synthesis algorithm that exploits ordering not only on the state but also on the input space. The synthesis of the maximal safety controller is done in two steps. First, we use only inputs with lower priorities to compute the maximal safety controller's domain $\mathrm{dom}(C^*)$. Then we synthesize the maximal controller by exploiting the inputs priorities, which makes it possible to compute the maximal safety controller without exploring all the inputs. We start by providing some characterizations of the maximal safety controller for LISM transition systems:

Proposition 3 *Consider a LISM transition system $T = (X, U, \Delta)$. Let C^* be the maximal safety controller enforcing the lower-closed safety specification $X^S \subseteq X$. The following properties hold:*

(i) $dom(C^*)$ is lower-closed w.r.t the partial order $\leq_X$;
(ii) for all $x_1, x_2 \in X$, if $x_1 \leq_X x_2$ then $C^*(x_2) \subseteq C^*(x_1)$;
(iii) for all $x \in X$, $C^*(x)$ is a lower-closed set w.r.t the partial order $\leq_U$;

PROOF. $(i), (ii)$ The result follows immediately from Proposition 2 and the fact that any LISM transition system is a LSM transition system.

(iii) Let $x \in X$, $u \in C^*(x)$ and $u' \in{\downarrow} u$. We have that $\Delta(x, u') \subseteq{\downarrow} \Delta(x, u) \subseteq{\downarrow} \mathrm{dom}(C^*) = \mathrm{dom}(C^*)$, where the first inclusion comes from the fact that T is a LISM transition system, the second inclusion comes from the fact that C^* is a safety controller and the last equality comes from the lower-closedness of $\mathrm{dom}(C^*)$. Hence, we have $\Delta(x, u') \subseteq \mathrm{dom}(C^*)$. Then, by maximality of C^*, $u' \in C^*(x)$. □

5.2.1 Domain of the controller

Let us define the set $U_{min} = \min(U)$ with respect to partial order $\leq_U$ on the input set U. Then the following is true.

Theorem 4 *Let C^* be the maximal safety controller for an LISM transition system $T = (X, U, \Delta)$ and lower-closed safety specification $X^S \subseteq X$. Let C_r^* be the maximal safety controller for the transition system $T_r = (X, U_{min}, \Delta)$ and safety specification X^S. Then we have $dom(C^*) = dom(C_r^*)$.*

PROOF. Let us define the controller C_r of the reduced transition system T_r and the safe set X^S as follows: for $x \in X$, $C_r(x) = C^*(x) \cap U_{min}$. First let us prove that $\mathrm{dom}(C_r) = \mathrm{dom}(C^*)$. The inclusion $\mathrm{dom}(C_r) \subseteq \mathrm{dom}(C^*)$ follows immediately from the construction of the controller C_r. Now let $x \in \mathrm{dom}(C^*)$ and let $u \in C^*(x)$. From (iii) in Proposition 2 we have that ${\downarrow} u \subseteq C^*(x)$, then there exists $u' \in U_{min}$ such that $u' \in C^*(x)$. Then, $u' \in C_r(x)$. Hence, $x \in \mathrm{dom}(C_r)$ and $\mathrm{dom}(C_r) = \mathrm{dom}(C^*)$ and C_r is a safety controller for T_r with a specification X^s. Now let us prove that for all $x \in X$, $C_r(x) = C_r^*(x)$. The first inclusion $C_r(x) \subseteq C_r^*(x)$ follows from maximality of the controller C_r^*. For the second inclusion, we have from maximality of C^* and since $U_{min} \subseteq U$ that $C_r^*(x) \subseteq C^*(x)$ for all $x \in X$. Moreover, by construction of C_r^*, we have that $C_r^*(x) \subseteq U_{min}$ for all $x \in X$. Then, $C_r(x) = C_r^*(x)$ for all $x \in X$. Since $\mathrm{dom}(C_r) = \mathrm{dom}(C^*)$, we have that $\mathrm{dom}(C_r^*) = \mathrm{dom}(C^*)$. □

The previous result states that for computation of the domain of the maximal safety controller $\mathrm{dom}(C^*)$, it is sufficient to use inputs with lower priorities.

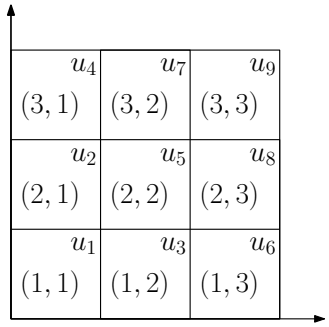


Fig. 4. Illustration of the reordering $U = \{u_1, \dots, u_9\}$ of the input set $U = \{1, 2, 3\}^2$ with respect to the component-wise partial order $\leq$ defined on $\mathbb{R}^2$.

Algorithm 3 Maximal Safety Controller

Input: LISM transition system $T = (X, U, \Delta)$, the domain of maximal safety controller $\mathrm{dom}(C^*)$.
Output: Controller C.

```
1  begin
2  |  for s ∈ X
3  |  |  C(s) := ∅;
4  |  for i = 1 : N
5  |  |  if u_i ∈ U_min then
6  |  |  |  S_i = dom(C*);
7  |  |  |  K_i = Pre(S_i, u_i, dom(C*));
8  |  |  else
9  |  |  |  S_i = ∩_{u_j <_U u_i} K_j;
10 |  |  |  K_i = Pre(S_i, u_i, dom(C*));
11 |  |  for s ∈ K_i
12 |  |  |  C(s) := C(s) ∪ {u_i};
13 |  return C
```

5.2.2 Maximal safety controller

It is always possible to reorder the elements of the input set $U = \{u_1, \dots, u_N\}$ as follows: for all $1 \leq j \leq i \leq N$ we suppose that either $u_j \leq_U u_i$ or $(u_i, u_j) \in \mathrm{Inc}_U$ (see Fig. 4 for the illustration). In this section, we exploit such an order to make synthesis for LISM transition systems more efficient. Algorithm 3 is based on the fact, that if a state is uncontrollable with an input $u \in U$, it is also uncontrollable with all the inputs $u' \in U$ such that $u' >_U u$, so if we failed to control a state with u, we do not need to explore $u' >_U u$ for it.

Let us now formally prove the result.

Lemma 3 *At every iteration of the loop 4-11 of the Algorithm 3 the set $K_i = Pre(S_i, u_i, dom(C^*))$ coincides with the set $Z_i = Pre(dom(C^*), u_i, dom(C^*))$.*

PROOF. To prove the result, we proceed by induction. For all i such that $u_i \in U_{min}$ the statement is obvious and we have the base. Let i be such that $u_i \notin U_{min}$. Suppose that for all $j < i$, $Z_j = K_j$, and let us prove that $Z_i = K_i$. Indeed, since for all $j < i$, $K_j = Z_j$ we have that $K_j \subseteq \mathrm{dom}(C^*)$ for all $j < i$, and as a consequence $S_i \subseteq \mathrm{dom}(C^*)$ (see line 8). From where $K_i \subseteq Z_i$ is immediately satisfied. At the same time, for all $x \in Z_i$ we have that $x \in \mathrm{dom}(C^*)$ and $\Delta(x, u_i) \subseteq \mathrm{dom}(C^*)$. Then from Proposition 3 we have that $x \in \mathrm{dom}(C^*)$ and for all $u_j <_U u_i$, $\Delta(x, u_j) \subseteq \mathrm{dom}(C^*)$. Consequently $x \in \mathrm{Pre}(\mathrm{dom}(C^*), u_j, \mathrm{dom}(C^*)) = Z_j = K_j$ for all j such that $u_j <_U u_i$. Then from the line 9 of Algorithm 3, we have that $x \in S_i$ and, as a consequence, from line 10, $x \in K_i$. $\square$

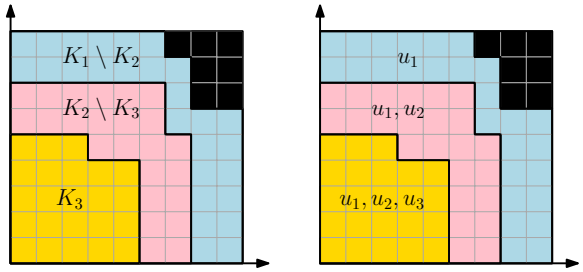


Fig. 5. Illustration of the maximal safety controller C^* for the case of a total order on the input set. $U = \{u_1, u_2, u_3\}$ with $u_1 \leq_U u_2 \leq_U u_3$.

Theorem 5 *Let $T = (X, U, \Delta)$ be a LISM transition system and let $dom(C^*)$ be a domain of the maximal safety controller C^* for a lower-closed safety specification $X^s \subseteq X$. Algorithm 3 returns the maximal safety controller C^*.*

PROOF. The statement follows immediately from Lemma 3, Theorem 3 and the fact that $u_i \in C(s)$ if and only if $s \in K_i$ (see lines 11-12 of the Algorithm 3).

Remark 1 *In the case of a total order*[2] *on the input set U, the set $K_i \subseteq K_j$ for all $N \geq j \geq i \geq 1$ (see Fig. 5).*

5.2.3 On memory requirements to implement C^*:

In symbolic control techniques, the symbolic controller is commonly represented as a look-up table ([27–30]), i.e, for every state controller stores the set of all possible control inputs that can be applied. Consequently, symbolic controllers are quite memory-consuming, which is a problem since they need to run on embedded devices with limited memory.

Some approaches have been proposed in the literature to simplify the symbolic controller representation such as Binary Decision Diagrams [31], Algebraic Decision Diagrams [32] and Decision Trees [33].

[2] A binary relation $\leq_{\mathcal{L}} \subseteq \mathcal{L} \times \mathcal{L}$ is a total order if it is a partial order and for all $l_1, l_2 \in \mathcal{L}$ we have either $l_1 \leq_{\mathcal{L}} l_2$ or $l_2 \leq_{\mathcal{L}} l_1$.

For LISM transition systems and directed safety specifications, one can use the monotonicity property to represent the maximal safety controller C^* efficiently. Indeed, we have from Proposition 2 that for a controllable state $x \in \mathrm{dom}(C^*)$, $C^*(x)$ is a lower-closed set. Hence, for the state x, instead of storing all the admissible control inputs $u \in C^*(x)$, one can only store the elements of the basis of $C^*(x)$, which allows a high reduction of the memory required to store the controller without losing its maximality. (cf. numerical examples).

6 Controller synthesis for intersections of upper and lower-closed safety specifications

In the previous sections, we have presented lazy synthesis algorithms to deal with lower (input-)state monotone transitions systems and lower-closed safety specifications. Following the duality between upper and lower-closed sets, the results for upper-closed safety specifications can be obtained using the same approaches.

This part aims to deal with more complex specifications described as intersections of upper and lower-closed sets. Let us mention that from Proposition 1, the intersection of lower (respectively upper) closed sets is again a lower (respectively upper) closed set. Hence, one can use the same approaches presented in the previous sections to deal with unions of lower (respectively upper) closed specifications.

In this section, we will mainly focus on the synthesis of controllers for the intersection of an upper and lower-closed set, which is a natural specification that frequently appears in control systems such as vehicle platoons [34], microgrids [20], trafic networks [35] and temperature regulation systems [14]. The considered setup is the following. Given a (input-)state monotone transition system $T = (X, U, \Delta)$, a lower-closed set $X^L \subseteq X$ and an upper-closed set $X^U \subseteq X$ with $X^L \cap X^U \neq \emptyset$, let us consider the problem of synthesizing the maximal safety controller for the transition system T and safety specification $X^{LU} = X^L \cap X^U$. We first have the following preliminary result

Lemma 4 *Consider the (input-)state monotone transition $T = (X, U, \Delta)$, the lower-closed $X^L \subseteq X$ and the upper-closed set $X^U \subseteq X$ with $X^L \cap X^U \neq \emptyset$. Let us define the following safety controllers:*

- *C^*_{LU} is the maximal safety controller for the transition system T and the safety specification $X^{LU} = X^L \cap X^U$;*
- *C^*_L is the maximal safety controller for the transition system T and the lower-closed safety specification X^L;*
- *C^*_U is the maximal safety controller for the transition system T and the upper-closed safety specification X^U.*

Then, for all $x \in X$, $C^*_{LU}(x) \subseteq C^*_L(x) \cap C^*_U(x)$.

PROOF. Since $X^{LU} \subseteq X^L$, we have that $C^*_{LU}(x) \subseteq C^*_L(x)$ for all $x \in X$. Similarly, using the fact that $X^{LU} \subseteq X^U$ we have that $C^*_{LU}(x) \subseteq C^*_U(x)$ for all $x \in X$. Hence, we have that $C^*_{LU}(x) \subseteq C^*_L(x) \cap C^*_U(x)$ for all $x \in X$. □

In the classical computation of the maximal safety controller C^*_{LU} defined above, we use the fixed-point-based approach defined in Algorithm 2, starting from the set $X^{LU} = X^L \cap X^U$ while exploring all the inputs $u \in U$. The idea here is to exploit the monotonicity property to incrementally synthesize the controller C^*_{LU} by proceeding in two steps:

(1) We synthesize the controllers C^*_U and C^*_L,
(2) We synthesize the maximal safety controller for the transition system T and safety specification $\mathrm{dom}(C^*_L) \cap \mathrm{dom}(C^*_U)$, where for each state $x \in \mathrm{dom}(C^*_L) \cap \mathrm{dom}(C^*_U)$, we explore only the inputs $u \in C^*_L(x) \cap C^*_U(x)$.

The completeness of the proposed incremental synthesis with respect to the direct synthesis is shown in the following result.

Proposition 4 *Under the preliminaries of Lemma 4, let us define the set $Z = dom(C^*_L) \cap dom(C^*_U)$. Let C^*_Z be the maximal safety controller for the transition system T and safety specifications Z. Then, for all $x \in X$, $C^*_{LU}(x) = C^*_Z(x)$.*

PROOF. Using the fact that $Z = \mathrm{dom}(C^*_L) \cap \mathrm{dom}(C^*_U) \subseteq X^{LU} = X^L \cap X^U$, it follows that $C^*_Z(x) \subseteq C^*_{LU}(x)$ for all $x \in X$. On the other hand, we have from Lemma 4 that $\mathrm{dom}(C^*_{LU}) \subseteq \mathrm{dom}(C^*_L) \cap \mathrm{dom}(C^*_U) = Z$. Hence, $C^*_{LU}(x) = C^*_Z(x)$ for all $x \in X$, which ends the proof. □

In some particular cases, one can obtain the maximal safety controller C^*_{LU} directly from the controllers C^*_L and C^*_U, without using the incremental synthesis, which is shown in the following result. (cf. numerical examples).

Proposition 5 *Under the preliminaries of Lemma 4, if for all $x \in dom(C^*_L) \cap dom(C^*_U)$, $C^*_L(x) \cap C^*_U(x) \neq \emptyset$, then for all $x \in X$, $C^*_{LU}(x) = C^*_L(x) \cap C^*_U(x)$.*

PROOF. The first inclusion follows from Lemma 4. To deal with the second inclusion, let us show that the controller $C_{LU} = C^*_L \cap C^*_U$ is a safety controller for the transition system T and safety specification X^{LU}. First, we have that $\mathrm{dom}(C_{LU}) \subseteq \mathrm{dom}(C^*_L) \cap \mathrm{dom}(C^*_U) \subseteq X^L \cap X^U = X^{LU}$. Hence, the first condition of Definition 6 is satisfied. Now let $x \in \mathrm{dom}(C_{LU})$ and $u \in$

$C_{LU}(x) = C_L^*(x) \cap C_U^*(x)$, the existence of such u is guaranteed by the fact that $C_L^*(x) \cap C_U^*(x) \neq \emptyset$. Since C_L^* and C_U^* are safety controllers, we have that $\Delta(x,u) \subseteq \mathrm{dom}(C_L^*) \cap \mathrm{dom}(C_U^*) = \mathrm{dom}(C_{LU})$. Hence, the second condition of Definition 6 is satisfied. Then, C^{LU} is a safety controller for the transition system T and safety specification X^{LU}. Hence, from maximality of the controller C_{LU}^*, we have that $C_{LU}(x) = C_L^*(x) \cap C_U^*(x) \subseteq C_{LU}^*(x)$, for all $x \in X$, which ends the proof. □

7 Abstractions for monotone dynamical systems

In this section, we first define the class of monotone dynamical systems. We then present different types of abstractions, namely box, and sparse abstractions. Finally, we show how to construct these abstractions to preserve monotonicity.

7.1 Discrete-time control systems

Let us start with a definition.

Definition 7 *A discrete-time control system Σ is a tuple $\Sigma = (X, U, D, f)$, where X is a set of states, U is a set of control inputs and D is a set of disturbance inputs. The function $f : X \times U \times D \longrightarrow X$ is called the transition function.*

Consider the discrete-time control system Σ of the form:

$$x(k+1) = f(x(k), u(k), d(k)),\ x(0) \in X \qquad (4)$$

where $x(k) \in X \subset \mathbb{R}^n$ is a state, $u(k) \in U \subset \mathbb{R}^m$ is a control input and $d(k) \in D \subseteq \mathbb{R}^p$ is a disturbance input. In the following we assume that the set of disturbances D is equipped with a partial order $\leq_D$ and described as a finite union of intervals $D = \bigcup_{m=1}^{M} [d_1^m, d_2^m]_D$. The discrete-time control system Σ is said to be state monotone (SM) if the set of states is equipped with a partial order and for all $x_1, x_2 \in X$, for all $u \in U$ and for all $d_1, d_2 \in D$, if $x_1 \leq_X x_2$ and $d_1 \leq_D d_2$ then $f(x_1, u, d_1) \leq_X f(x_2, u, d_2)$. Similarly the discrete-time control system Σ is said to be input-state monotone (ISM) if its sets of states and inputs are equipped with partial orders and for all $x_1, x_2 \in X,\ u_1, u_2 \in U$ and for all $d_1, d_2 \in D$, if $x_1 \leq_X x_2$, $u_1 \leq_U u_2$ and $d_1 \leq_D d_2$ then $f(x_1, u_1, d_1) \leq_X f(x_2, u_2, d_2)$.

7.2 Box abstraction

In this part, we construct a symbolic box abstraction $T_d^B(\Sigma) = (X_d, U_d, \Delta_d)$ for the original system $\Sigma = (X, X, U, D, f)$. Then, we show that using such construction, monotonicity property is preserved when going from the original system to its symbolic abstraction.

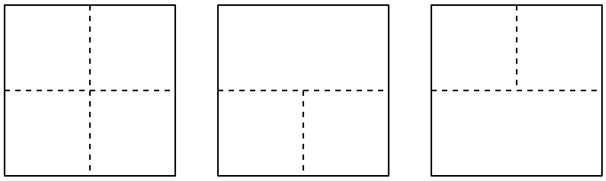


Fig. 6. The first two partitions satisfy Assumption 1, while the third partition does not satisfy Assumption 1.The first and third partitions satisfy Assumption 2, while the second partition does not satisfy Assumption 2. The state-space is equipped with the component-wise partial order $\leq$ defined on $\mathbb{R}^2$

7.2.1 Discretization

The construction of the symbolic box abstraction $T_d^B(\Sigma)$ is based on a discretization of the state-space and input sets. We approximate the set of inputs U with a finite number of values n_u: $U_d = \left\{ \mathsf{u}_\ell \in U |\ \ell = 0, \ldots, n_u - 1 \right\}$. We discretize the state-space into $n_x \geq 1$ states using a finite partition [3] X_d of the set X. Each element q of the partition can be described as an interval $q = [x_1^q, x_2^q]_X$. We define the quantizer $Q_{X_d} : X \to X_d$ associated to the partition X_d as follows: for $x \in X$ and $q \in X_d$, $Q_{X_d}(x) = q$ if and only if $x \in q$. We make the following assumptions on the discrete states of the set X_d.

Assumption 1 *For all $q, q' \in X_d$ if there exists $(x, x') \in q \times q'$ satisfying $x \leq_X x'$, then $x_2^q \leq_X x_2^{q'}$.*

Assumption 2 *For all $q, q' \in X_d$ if there exists $(x, x') \in q \times q'$ satisfying $x \leq_X x'$, then $x_1^q \leq_X x_1^{q'}$.*

Intuitively, Assumption 1 (respectively, Assumption 2) reflects the fact that the quantizer should preserve the lower (respectively, upper) monotonicity property from continuous to discrete (symbolic) states. Fig. 6 shows examples of partitions satisfying Assumptions 1 and 2.

Keeping in mind Assumptions 1 and 2, we define a partial order $\leq_{X_d}$ over the set of discrete states X_d as follows:

- If Assumption 1 is satisfied, then for all $q_1, q_2 \in X_d$, $q_1 \leq_{X_d} q_2$ if and only if $x_2^{q_1} \leq_X x_2^{q_2}$;
- If Assumption 2 is satisfied, then for all $q_1, q_2 \in X_d$, $q_1 \leq_{X_d} q_2$ if and only if $x_1^{q_1} \leq_X x_1^{q_2}$;
- If Assumptions 1 and 2 are satisfied, then for all $q_1, q_2 \in X_d$, $q_1 \leq_{X_d} q_2$ if and only if $x_1^{q_1} \leq_X x_1^{q_2}$ and $x_2^{q_1} \leq_X x_2^{q_2}$.

7.2.2 Transition relation

Since we work with (input-)state-monotone discrete control systems [38], we can link the original system Σ and

[3] In order to define a partition, the sets of measure zero where intervals overlap can be ignored for notational convenience, (see e.g. [36,37])

its symbolic box abstraction $T_d^B(\Sigma)$ with feedback refinement relation [39], by defining the transition relation $\Delta_d \subseteq X_d \times U_d \times X_d$, as follows: for $q \in X_d$, $u \in U_d$, $q' \in \Delta_d(q,u)$ if and only if there exists $m \in \{1,\dots,M\}$ such that $[f(x_1^q,u,d_1^m), f(x_2^q,u,d_2^m)]_X \cap [x_1^{q'}, x_2^{q'}]_X \neq \emptyset$.

We illustrate the construction of the transitions for the box abstraction in Figure 7. In the following result, we show that, under Assumptions 1 and 2, monotonicity of the discrete-time control system Σ is preserved when constructing its symbolic box abstraction $T_d^B(\Sigma)$.

Proposition 6 *Let us consider the discrete-time control system $\Sigma = (X, X, U, D, f)$. If Σ is an ISM system and if its symbolic box abstraction $T_d^B(\Sigma)$ satisfies Assumption 1, (respectively, Assumption 2), then $T_d^B(\Sigma)$ is a LISM (respectively, UISM) transition system. Moreover, when both Assumptions 1 and 2 are satisfied, then $T_d^B(\Sigma)$ is an ISM transition system*

PROOF. We only provide a proof for the case of LISM, the cases of UISM and ISM can be derived similarly. We should prove that for all $q_1, q_2 \in X_d$ and for all $u_1, u_2 \in U_d$, if $q_1 \leq_{X_d} q_2$ and $u_1 \leq_{U_d} u_2$, the following is satisfied: for all $q_1' \in \Delta_d(q_1,u_1)$ there is $q_2' \in \Delta_d(q_2,u_2)$, such that $q_1' \leq_{X_d} q_2'$.

Let $q_1, q_2 \in X_d$ and $u_1, u_2 \in U_d$, such that $q_1 \leq_{X_d} q_2$ and $u_1 \leq_{U_d} u_2$. Under Assumption 1, the last is equivalent to $x_2^{q_1} \leq_{X_d} x_2^{q_2}$ and $u_1 \leq_{U_d} u_2$. Let $q_1' \in \Delta_d(q_1,u_1)$, from the monotonicity of Σ and the construction of $T_d^B(\Sigma)$, we have the existence of $m \in \{1,\dots,M\}$ such that $f(x_2^{q_1}, u_1, d_2^m) \in q_1'$. Let $q_2' \in X_d$ is such that $f(x_2^{q_2}, u_2, d_2^m) \in q_2'$. Then $q_2' \in \Delta_d(q_2,u_2)$. Moreover, there are $\underline{x} = f(x_2^{q_1}, u_1, d_2^m)$ and $\overline{x} = f(x_2^{q_2}, u_2, d_2^m)$, such that $\underline{x} \in q_1', \overline{x} \in q_2'$ and since Σ is an input-state monotone control system, one has $\underline{x} \leq_X \overline{x}$. Hence, from Assumption 1, $x_2^{q_1'} \leq_X x_2^{q_2'}$. The last is equivalent to $q_1' \leq_{X_d} q_2'$. □

We then have the following corollary for SM systems.

Corollary 2 *Let us consider the discrete-time control system $\Sigma = (X, U, D, f)$. If Σ is a SM system and if its symbolic box abstraction $T_d^B(\Sigma)$ satisfies Assumption 1, (respectively, Assumption 2), then $T_d^B(\Sigma)$ is a LSM (respectively, USM) transition system. Moreover, when both Assumptions 1 and 2 are satisfied, then $T_d^B(\Sigma)$ is a SM transition system.*

Hence, the monotonicity property is preserved when going from the original system to its symbolic abstraction. Let $X^s \subseteq X$ be a safe set and let us define a safety specification for the symbolic abstraction $T_d^B(\Sigma)$. We say that a state $q \in X_d$ belongs to a safe set $X_d^s \subseteq X_d$ if and only if $q \subseteq X^s$. In this paper we consider directed (upper and lower closed) safety specifications. If the set X^s is lower (respectively, upper) closed with respect to the partial order $\leq_X$ and Assumption 1 (respectively, Assumption 2) holds, then the set X_d^s is lower (respectively, upper) closed with respect to the partial order $\leq_{X_d}$. So, we can use Algorithm 1, Algorithm 2 and Algorithm 3 to find a controller for the abstraction. Then, since we link the system Σ and its symbolic abstraction $T_d^B(\Sigma)$ with feedback refinement relation, we can refine a safety controller for the transition system $T_d^B(\Sigma)$ to a controller to the original system Σ [39]. It is also, noteworthy to remark that the abstraction can be computed on fly, while executing the synthesis algorithms and only in case of demand.

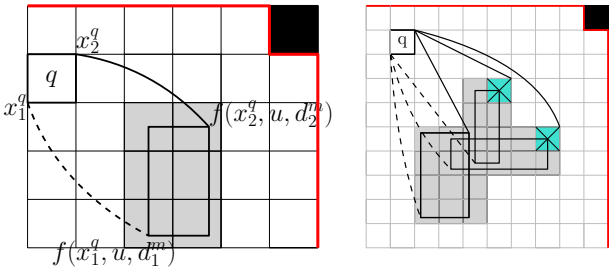


Fig. 7. Left: A box abstraction for a fixed disturbance interval $[d_1^m, d_2^m]$ $m \in \{1,\dots,M\}$. Right: box (grey) and sparse (blue) abstractions for a monotone system with a lower–closed safety specification for $M = 3$.

7.3 Sparse abstraction

For the discrete-time control system $\Sigma = (X, U, D, f)$ introduced in Definition 7, an upper-sparse abstraction is defined as $T_d^{US}(\Sigma) = (X_d, U_d, \Delta_d^{US})$ where X_d, U_d are inherited from $T_d^B(\Sigma)$ and the transition relation is defined for $q \in X_d$, $u \in U_d$ as $\Delta_d^{US}(q,u) = \max(\Delta_d(q,u))$. An illustration of the construction of an upper sparse abstraction is shown in Figure 7. Similarly, a lower-sparse abstraction is defined as $T_d^{LS}(\Sigma) = (X_d, U_d, \Delta_d^{LS})$ where X_d, U_d are inherited from $T_d^B(\Sigma)$ and the transition relation is defined for $q \in X_d$, $u \in U_d$ as $\Delta_d^{US}(q,u) = \min(\Delta_d(q,u))$.

Remark 2 *Let us remark that the transition relation of the upper sparse abstraction can be equivalently defined as follows: for all $q \in X_d, u \in U_d$, $q' \in \Delta_d^{US}$ if and only $q' \in \max(\cup_{m=1}^M q^m)$, where q^m is such that $f(x_2^q, u, d_2^m) \in q^m$ (see Figure 7 for an illustration).*

Although, our definition of upper-sparse abstraction is slightly different from what have been proposed in [21], the transition system $T_d^{US}(\Sigma)$ is still related to $T_d^B(\Sigma)$ with an upper alternating simulation relation [21].

We first show how to preserve the monotonicity when constructing upper or lower-sparse abstractions.

Proposition 7 *Under Assumption 1, if the box abstraction $T_d^B(\Sigma)$ of the discrete-time control system $\Sigma = (X, U, D, f)$ is LISM, then its upper-sparse abstraction $T_d^{US}(\Sigma)$ is also LISM. Similarly, under Assumption 2, if the box abstraction $T_d^B(\Sigma)$ of the discrete-time control system $\Sigma = (X, U, D, f)$ is UISM, then its lower-sparse abstraction $T_d^{US}(\Sigma)$ is also UISM.*

PROOF. We only provide a proof for the case of LISM, the case of UISM can be derived similarly. Consider $q_1, q_2 \in X_d$, $u_1, u_2 \in U_d$ such that $q_1 \leq_{X_d} q_2$, $u_1 \leq_{U_d} u_2$ and let $q_1' \in \Delta_d^{US}(q_1, u_1) = \max(\Delta_d(q_1, u_1))$, then $q_1' \in \Delta_d(q_1, u_1)$. From Proposition 6, $T_d^B(\Sigma)$ is input-state monotone, then there exists $q_2' \in \Delta_d(q_2, u_2)$ such that $q_1' \leq_{X_d} q_2'$. Coupling the last with the fact that there exists $\overline{q}_2' \in \max(\Delta_d(q_2, u_2))$ such that $q_2' \leq_{X_d} \overline{q}_2'$ (see Definition 1), we finally get that $T_d^{US}(\Sigma)$ is ISM. □

We then have the following corollary for SM systems.

Corollary 3 *Under Assumption 1, if the box abstraction $T_d^B(\Sigma)$ of the discrete-time control system $\Sigma = (X, U, D, f)$ is LSM, then its upper-sparse abstraction $T_d^{US}(\Sigma)$ is also LSM. Similarly, under Assumption 2, if the box abstraction $T_d^B(\Sigma)$ of the discrete-time control system $\Sigma = (X, U, D, f)$ is USM, then its lower-sparse abstraction $T_d^{US}(\Sigma)$ is also USM.*

We now show the equivalence between box and upper-sparse (respectively, lower-sparse) abstractions when dealing with lower-closed (respectively, upper-closed) safety specifications.

Proposition 8 *Consider the ISM discrete-time control system $\Sigma = (X, U, D, f)$ introduced in Definition 7. If X_d^s is a lower-closed safety specification, then the maximal safety controller C_B^* for the box abstraction $T_d^B(\Sigma)$ and the safety specification X_d^s coincides with the maximal safety controller C_{US}^* for the upper-sparse abstraction $T_d^{US}(\Sigma)$ and the safety specification X_d^s. Similarly, if X_d^s is an upper-closed safety specification, then the maximal safety controller C_B^* for the box abstraction $T_d^B(\Sigma)$ and the safety specification X_d^s coincides with the maximal safety controller C_{LS}^* for the lower-sparse abstraction $T_d^{LS}(\Sigma)$ and the safety specification X_d^s.*

PROOF. We only provide a proof for the case of lower-closed safety specifications, the case of upper-closed safety specifications can be derived similarly. Let us show that C_B^* is a safety controller for $T_d^{US}(\Sigma)$. Indeed, since C_B^* is a safety controller for T_d^B it is obvious that $\mathrm{dom}(C_B^*) \subseteq X_d^s$. Let $q \in \mathrm{dom}(C_B^*)$, then for all $u \in C_B^*(q)$ the following is satisfied $\Delta_d^{US}(q, u) \subseteq \Delta_d(q, u) \subseteq \mathrm{dom}(C_B^*)$. Hence, C_B^* is a safety controller for $T_d^{US}(\Sigma)$ and safe set X_d^s. Then, from the definition of maximal safety controller one has $C_B^*(q) \subseteq C_{US}^*(q)$ for all $q \in X_d$.

Let us show that C_{US}^* is a safety controller for $T_d^B(\Sigma)$. Again, it is obvious, that $\mathrm{dom}(C_{US}^*) \subset X_d^s$. Let $q \in \mathrm{dom}(C_{US}^*), u \in C_{US}^*(q)$ and let $q' \in \Delta_d(q, u)$. Then there exists $\overline{q}' \in \Delta_d^{US}(q, u) = \max(\Delta_d(q, u))$ such that $q' \leq_{X_d} \overline{q}'$. Since $\Delta_d^{US}(q, u) \subseteq \mathrm{dom}(C_{US}^*)$ we have that $\overline{q}' \in \mathrm{dom}(C_{US}^*)$. Moreover, since the safe set X_d^s is lower-closed, we have from Proposition 3 that $\mathrm{dom}(C_{US}^*)$ is lower-closed and one we get that $q' \in \mathrm{dom}(C_{US}^*)$. Hence, C_{US}^* is a safety controller for $T_d^B(\Sigma)$ and from the maximality of the controller C_B^*, one has $C_{US}^*(q) \subseteq C_B^*(q)$ for all $q \in X_d$. □

We then have the following corollary for SM systems.

Corollary 4 *Consider the SM discrete-time control system $\Sigma = (X, U, D, f)$ introduced in Definition 7. If X_d^s is a lower-closed safety specification, then the maximal safety controller C_B^* for the box abstraction $T_d^B(\Sigma)$ and the safety specification X_d^s coincides with the maximal safety controller C_{US}^* for the upper-sparse abstraction $T_d^{US}(\Sigma)$ and the safety specification X_d^s. Similarly, if X_d^s is an upper-closed safety specification, then the maximal safety controller C_B^* for the box abstraction $T_d^B(\Sigma)$ and the safety specification X_d^s coincides with the maximal safety controller C_{LS}^* for the lower-sparse abstraction $T_d^{LS}(\Sigma)$ and the safety specification X_d^s.*

Intuitively, the result of Proposition 8 shows that we can use sparse abstractions instead of box abstraction while working with (input-) state monotone systems and directed specifications.

Remark 3 *Let us mention that for the proposed incremental safety synthesis procedure in Section 6, where the safe set is given as an intersection of an upper and lower closed sets, we use the different types of abstractions presented in this section. Indeed, in the first step we use an upper sparse abstraction to compute a controller for the lower-closed set. Similarly, we use a lower sparse abstraction to compute the controller for the upper-closed set. Finally, we use the box abstraction while using the previously obtained controllers as a warm point, which allows us to speed-up the computations.*

8 Numerical example

8.1 Model description

As an example, we consider two vehicles moving along a straight road. Each vehicle is modeled as a point mass m with velocity changing according to the law

$$m\dot{v} = \alpha(F, v) = F - (f_0 + f_1 v + f_2 v^2). \qquad (5)$$

In the equation above, F represents a net action of braking and engine torque applied to the wheels, while the second term $f_0 + f_1 v + f_2 v^2$ describes aerodynamic and rolling resistance effects. The net force F is viewed as a control input u for the follower vehicle and a disturbance for the lead one. It is assumed to be bounded by

$$F_{min} = -0.3mg \leq F \leq 0.2mg = F_{max},$$

where g is a gravitational constant. Such a bound is consistent with non-emergency braking and acceleration.

We slightly adapt equation (5) to prohibit a back motion for the follower:

$$m\dot{v} = \begin{cases} \alpha(u, v) & \text{if } \ v > 0 \\ \max(u - f_0, 0) & \text{if } \ v = 0 \end{cases} \tag{6}$$

For the leader we assume that its velocity w remains in a range $[0, w_{max}]$:

$$m\dot{w} = \begin{cases} \alpha(a, w) & \text{if } \ 0 < w < w_{max} \\ \max(\alpha(a, w), 0) & \text{if } \ w = 0 \\ \min(\alpha(a, w), 0) & \text{if } \ w = w_{max} \end{cases} \tag{7}$$

Combining (6) and (7) with the equation

$$\dot{d} = w - v, \tag{8}$$

which describes the distance between two vehicles, we obtain the final model of the system.

The values of parameters shown in Table 1 are taken from [40]. In the following, the implementation has been done in MATLAB, Processor Intel Core i7-4870HQ, 2,5 GHz, RAM 16 GB.

8.2 Control objective and numerical results

8.2.1 Lower-closed safety specification

We start with the objective to synthesize a controller for the follower vehicle, to keep its velocity below $v_{\max} = 30\,m/s$ and to guarantee that the relative distance between the leader and the follower remains larger than $d_{\min} = 10\,m$, while assuming that the leader car acts as a disturbance $a \in [F_{min}, F_{max}]$ and $w_{max} = 30\,m/s$. The following change of coordinates: $h = -d$, $z = -w$ transform the system (6)-(7)-(8) into a monotone one. Moreover, after this change of coordinates, we get a lower-closed safety specification.

Setting a time step $\tau = 0.8\,s$, we generate a discrete-time model corresponding to the continuous-time system (6)-(7)-(8). We also introduce a Cartesian partition on the

Table 1
Vehicle parameters

Parameter	Value	Unit
m	1370	Kg
f_0	51.0709	N
f_1	0.3494	Ns/m
f_2	0.4161	Ns^2/m^2

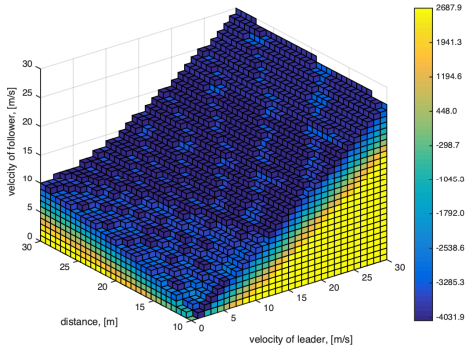


Fig. 8. Maximal safety controller C^*

state space X and input space U with $n_x = (31, 31, 31)$ and $n_u = 50$ as state and input discretization parameters correspondingly. We then construct a sparse symbolic model and use Algorithm 3 to synthesize the maximal safety controller C^* for the abstraction. Let us remind that if for a state $x \in X$ an input $u \in C^*(x)$, then all inputs satisfying $u' \leq_U u$ are enabled by C^* for this state. Moreover, in our example, we have total order on the input space U. Hence, for each state $x = (w, d, v)$, it is enough to store only the maximal safe input to reconstruct the whole maximal safety controller. Consequently, the amount of memory required for the controller implementation is significantly reduced.

We represent the obtained maximal safety controller C^* in Figure 8. The blue color of the color bar corresponds to the minimal input $F_{\min} = -4031.9\ mKg/s^2$ and the yellow color corresponds to the maximal input $F_{\max} = 2687.9\ mKg/s^2$. For a given state $x = (w, d, v)$, Figure 8 shows the maximal allowed control input.

To evaluate the performance of our approach, we compare it with the classical safety synthesis algorithm while exploring different scenarios: we compare the proposed approach in this paper with the box abstractions [1] and the sparse abstractions [21]. We also provide a comparison with lazy synthesis procedure, based on three valued abstractions [15]. In [15], once a safe input is found for a state, the other controls are not explored for this state. The authors start with an empty transition relation and iteratively add new transitions essential for the synthesis purpose. At every iteration, the precomputed part of the symbolic model is explored. Then, one randomly chooses $N_{3v} \in \mathbb{N}_{>0}$ states from uncontrollable states,

where N_{3v} is a parameter given by user. For each of chosen states, one randomly picks an unexplored input and adds the corresponding tradition to the abstraction. The algorithm returns only the domain of the maximal safety controller and not the whole maximal safety controller. Indeed, for every state, only one safe input is returned. Moreover, since the usage of sparse abstractions speed-up synthesis process we also use them when implementing the approach from [15].

The evaluation is based on two criteria, the computation time and the memory required to implement the maximal safety controller. We explore two different scenarios. In the first case, we vary the state-space discretization parameter n_x while keeping the input discretization parameter as a constant $n_u = 10$. The results of run time and memory comparison are represented in Table 2 and Table 3. In the second case, we set $n_x = (31, 31, 31)$ and vary the input discretization parameter n_u. The computational results are given in Table 4 and Table 5.

In Tables 2, 4, time T^s_{lm} is a running time of Algorithm 3. Time T_{cl} is a running time of the classical fixed point algorithm when box abstractions are used [1]. Time T^s_{cl} is a running time of the classical fixed point algorithm when sparse symbolic models are used [21]. Time T^s_{3v} is a running time of the lazy algorithm from [15] implemented for sparse abstractions. For the parameter N_{3v} the value $round(0.6 * n_x(1) * n_x(2) * n_x(3))$ have been chosen. We also store the amount of memory needed for controllers implementation in variables M^s_{lm}, M_{cl}, M_{cl} and M^s_{3v} correspondingly. Let us remind that for ISM with directed specifications the controllers synthesised for sparse abstractions and box abstractions coincide, hence $M_{cl} = M^s_{cl}$ (column 3 in Tables 3,5). However, it is obvious from columns 3 and 4 of Tables 2,4 that sparse abstractions-based synthesis is much faster, so we repeated the result from [21]. Since our controller and 3-valued abstractions-based controller store just one safe input for every state, their memory requirements also coincide (column 2 in Tables 3,5). However, we store the maximal safe input for every state. We can easily reconstruct the maximal safety controller without any computations. In contrast, the three-valued abstractions-based controller store just a random safe input, and to get the maximal safety controller, one should check for all the other inputs if they allow to remain in the controllable domain. The latter is not efficient when the set of discrete inputs is large.

The numerical results highlight the practical speedups and memory efficiency that can be attained using the lazy approach while ensuring completeness w.r.t the classical safety algorithm.

Table 2
Runtime comparison when varying the number of states. $T^s_{lm}, T_{cl}, T^s_{cl}$ and T^s_{3v} are the running time of Algorithm 3, the classical fixed point algorithm with box abstractions [1], the classical fixed point algorithm with sparse abstractions [21] and the lazy algorithm from [15], respectively.

n_x	T^s_{lm}	T_{cl}/T^s_{lm}	T^s_{cl}/T^s_{lm}	T^s_{3v}/T^s_{lm}
(15,15,15)	2.96 s	31.22	15.52	11.18
(31,31,31)	18.16 s	53.35	23.19	14.92
(63,63,63)	118.67 s	85.21	30.85	16.25

Table 3
Memory comparison when varying the number of states. $M^s_{lm}, M_{cl}, M^s_{cl}$ and M^s_{3v} are the required memory to implement the controller resulting from Algorithm 3, the classical fixed point algorithm with box abstractions [1], the classical fixed point algorithm with sparse abstractions [21] and the lazy algorithm from [15], respectively.

n_x	$M^s_{lm} = M^s_{3v}$	$M_{cl} = M^s_{cl}$	M_{cl}/M_{lm}
(15,15,15)	26.4 KB	499.9 KB	18.93
(31,31,31)	232.7 KB	4729 KB	20.32
(63,63,63)	1953.5 KB	41469.3 KB	21.22

8.2.2 Intersection of lower and upper-closed safety specifications

Let us consider another control objective to illustrate the results of Section 6. We want to synthesise a controller for the follower vehicle, which takes its values from $[F_{min}, F_{max}]$, to keep the velocity of the follower below $v_{\max}$ and guarantees that the relative distance between the leader and the follower remains larger than $d_{\min} = 10\,m$ and smaller than $d_{\max} = 150\,m$, while assuming that the leader vehicle acts as a disturbance $d \in [0.65 * F_{min}, 0.65 * F_{max}]$ and $w_{max} = 25\,m/s$. In this case, we can use the approach proposed in Proposition 4 to speed up the computation by a factor of 3.7 in comparison to the classical approach. Time gain is a result of using a sparse abstraction, instead of a more common box abstraction, to find C^*_L and C^*_U and then use them as a warm point for the following computations. Since our abstraction is input-state monotone and we have the total order on set of inputs, it is sufficient for every state x to store only safe actions $u^s_{min}(x),\ u^s_{max}(x)$ with minimal and maximal value, because all inputs in a range $[u^s_{min}(x), u^s_{max}(x)]$ are admissible. See the maximal controller in the Figure 9.

If we set up the parameter d_{max} as $200\,m$ the assumption of Proposition 5 is satisfied and $C^*_{LU}(x) = C^*_L(x) \cap C^*_U(x)$. In this case we can fully used the benefits of the Algorithm 3, and we are 45.7 times faster than the classical approach.

Table 4
Runtime comparison when varying the number of states. T^s_{lm}, T_{cl}, T^s_{cl} and T^s_{3v} are the running time of Algorithm 3, the classical fixed point algorithm with box abstractions [1], the classical fixed point algorithm with sparse abstractions [21] and the lazy algorithm from [15], respectively.

n_u	T^s_{lm}	T_{cl}/T^s_{lm}	T^s_{cl}/T^s_{lm}	T^s_{3v}/T^s_{lm}
10	18.19 s	54.01	23.42	15.06
20	20.56 s	95.08	41.08	25.96
40	25.26 s	154.03	66.67	43.25

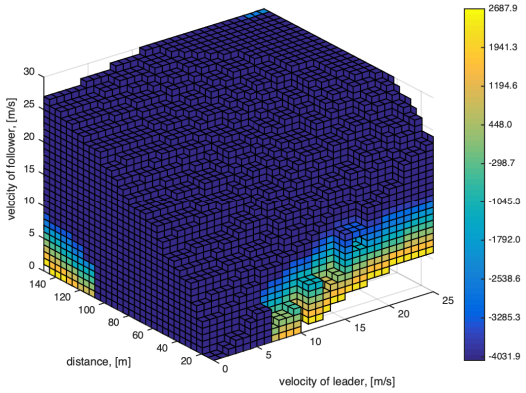
Table 5
Memory comparison when varying the number of states. M^s_{lm}, M_{cl}, M^s_{cl} and M^s_{3v} are the required memory to implement the controller resulting from Algorithm 3, the classical fixed point algorithm with box abstractions [1], the classical fixed point algorithm with sparse abstractions [21] and the lazy algorithm from [15], respectively.

n_u	$M^s_{lm} = M^s_{3v}$	$M_{cl} = M^s_{cl}$	M_{cl}/M_{lm}
10	232.7 KB	4729 KB	20.32
20	232.7 KB	6200.1 KB	26.64
40	232.7 KB	9141.8 KB	39.29

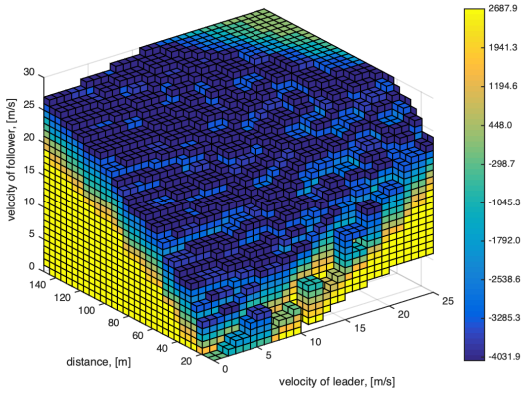
9 Conclusion

In this paper, we have presented an efficient approach to controller synthesis for monotone transition systems and directed safety specifications. First, we presented a lazy algorithm for the computation of predecessors, based on which lazy synthesis algorithms are proposed allowing us to explore ordering on the states (and inputs). Then an incremental controller synthesis approach is presented, allowing to deal with intersections of directed specifications. Numerical results highlight the practical speedups and memory efficiency that can be attained using the proposed approach while ensuring completeness w.r.t the classical safety algorithm.

In future work, we will develop more general algorithms allowing to extend the approach to other types of directed specifications, such as reachability, stability, or more general properties described by a temporal logic formula.



(a) Minimal value



(b) Maximal value

Fig. 9. Maximal safety controller C^*

References

[1] P. Tabuada, *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[2] C. Belta, B. Yordanov, and E. Gol, *Formal methods for discrete-time dynamical systems*. Springer, 2017.

[3] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems, 2nd ed.* Springer, 2009.

[4] A. Saoud, *Compositional and Efficient Controller Synthesis for Cyber-Physical Systems*. PhD thesis, Université Paris Saclay, 2019.

[5] P. Nilsson, *Correct-by-Construction Control Synthesis for High-Dimensional Systems*. PhD thesis, University of Michigan, 2017.

[6] A. Weber, M. Rungger, and G. Reissig, "Optimized state space grids for abstractions," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5816–5821, 2017.

[7] A. Saoud and A. Girard, "Optimal multirate sampling in symbolic models for incrementally stable switched systems," *Automatica*, vol. 98, pp. 58–65, 2018.

[8] A. Swikir and M. Zamani, "Compositional synthesis of finite abstractions for networks of systems: A small-gain approach," *Automatica*, vol. 107, pp. 551–561, 2019.

[9] P.-J. Meyer, A. Girard, and E. Witrant, "Compositional abstraction and safety synthesis using overlapping symbolic models," *IEEE Transactions on Automatic Control*, vol. 63, no. 6, pp. 1835–1841, 2017.

[10] A. Saoud, P. Jagtap, M. Zamani, and A. Girard, "Compositional abstraction-based synthesis for interconnected systems: An approximate composition approach," *IEEE Transactions on Control of Network Systems*, vol. 8, no. 2, pp. 702–712, 2021.

[11] E. A. Gol, M. Lazar, and C. Belta, "Language-guided controller synthesis for linear systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1163–1176, 2013.

[12] A. Girard, G. Gössler, and S. Mouelhi, "Safety controller synthesis for incrementally stable switched systems using multiscale symbolic models," *IEEE Transactions on Automatic Control*, vol. 61, no. 6, pp. 1537–1549, 2016.

[13] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck, "Lazy abstraction-based control for safety specifications," in *IEEE Conference on Decision and Control*, pp. 4902–4907, 2018.

[14] E. Ivanova and A. Girard, "Lazy safety controller synthesis with multi-scale adaptive-sampling abstractions of nonlinear systems," vol. 53, pp. 1837–1843, Elsevier, 2020.

[15] O. Hussien and P. Tabuada, "Lazy controller synthesis using three-valued abstractions for safety and reachability specifications," in *IEEE Conference on Decision and Control*, pp. 3567–3572, 2018.

[16] A. Kader, A. Saoud, and A. Girard, "Safety controller design for incrementally stable switched systems using event-based symbolic models," in *European Control Conference*, pp. 1269–1274, 2019.

[17] E. Ivanova and A. Girard, "Lazy symbolic controller for continuous-time systems based on safe set boundary exploration," in *7th IFAC Conference on Analysis and Design of Hybrid Systems*, 2021.

[18] S. Coogan, M. Arcak, and C. Belta, "Formal methods for control of traffic flow: Automated control synthesis from finite-state transition models," *IEEE Control Systems Magazine*, vol. 37, no. 2, pp. 109–128, 2017.

[19] D. Angeli and E. D. Sontag, "Monotone control systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 10, pp. 1684–1698, 2003.

[20] D. Zonetti, A. Saoud, A. Girard, and L. Fribourg, "Decentralized monotonicity-based voltage control of dc microgrids with zip loads," *IFAC-PapersOnLine*, vol. 52, no. 20, pp. 139–144, 2019.

[21] E. S. Kim, M. Arcak, and S. A. Seshia, "Symbolic control design for monotone systems with directed specifications," *Automatica*, vol. 83, pp. 10–19, 2017.

[22] S. Sadraddini and C. Belta, "Safety control of monotone systems with bounded uncertainties," in *IEEE Conference on Decision and Control*, pp. 4874–4879, 2016.

[23] A. Saoud, E. Ivanova, and A. Girard, "Efficient synthesis for monotone transition systems and directed safety specifications," in *IEEE Conference on Decision and Control*, pp. 6255–6260, 2019.

[24] U. S. Reddy, "Topology and stone duality for domains," 1995.

[25] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!," *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63–92, 2001.

[26] G. Higman, "Ordering by divisibility in abstract algebras," *Proceedings of the London Mathematical Society*, vol. 3, no. 1, pp. 326–336, 1952.

[27] M. Mazo, A. Davitian, and P. Tabuada, "Pessoa: A tool for embedded controller synthesis," in *International Conference on Computer Aided Verification*, pp. 566–569, Springer, 2010.

[28] S. Mouelhi, A. Girard, and G. Gössler, "Cosyma: a tool for controller synthesis using multi-scale abstractions," in *International Conference on Hybrid Systems: Computation and Control*, pp. 83–88, 2013.

[29] M. Rungger and M. Zamani, "Scots: A tool for the synthesis of symbolic controllers," in *International Conference on Hybrid Systems: Computation and Control*, pp. 99–104, 2016.

[30] M. Khaled and M. Zamani, "pfaces: an acceleration ecosystem for symbolic control," in *International Conference on Hybrid Systems: Computation and Control*, pp. 252–257, 2019.

[31] I. S. Zapreev, C. Verdier, and M. Mazo Jr, "Optimal symbolic controllers determinization for bdd storage," *IFAC Conference on Analysis and Design of Hybrid Systems*, vol. 51, no. 16, pp. 1–6, 2018.

[32] A. Girard, "Low-complexity quantized switching controllers using approximate bisimulation," *Nonlinear Analysis: Hybrid Systems*, vol. 10, pp. 34–44, 2013.

[33] P. Ashok, M. Jackermeier, P. Jagtap, J. Křetínskỳ, M. Weininger, and M. Zamani, "dtcontrol: Decision tree learning algorithms for controller representation," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pp. 1–7, 2020.

[34] P. Nilsson, O. Hussien, Y. Chen, A. Balkan, M. Rungger, A. Ames, J. Grizzle, N. Ozay, H. Peng, and P. Tabuada, "Preliminary results on correct-by-construction control software synthesis for adaptive cruise control," *IEEE Conference on Decision and Control*, pp. 816–823, 2014.

[35] S. Coogan, E. A. Gol, M. Arcak, and C. Belta, "Traffic network control from temporal logic specifications," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 2, pp. 162–172, 2015.

[36] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta, "Temporal logic control of discrete-time piecewise affine systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1491–1504, 2011.

[37] S. Coogan and M. Arcak, "Efficient finite abstraction of mixed monotone systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pp. 58–67, 2015.

[38] T. Moor and J. Raisch, "Abstraction based supervisory controller synthesis for high order monotone continuous systems," *Modelling, Analysis, and Design of Hybrid Systems, Springer*, pp. 247—-265, 2002.

[39] G. Reissig, A. Weber, and M. Rungger, "Feedback refinement relations for the synthesis of symbolic controllers," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1781–1796, 2017.

[40] A. Saoud, A. Girard, and L. Fribourg, "Contract-based design of symbolic controllers for safety in distributed multiperiodic sampled-data systems," *IEEE Transactions on Automatic Control*, vol. 66, no. 3, pp. 1055–1070, 2020.