Discrete Event Dyn Syst (2017) 27:301–340
DOI 10.1007/s10626-017-0243-z

Augmented finite transition systems as abstractions for control synthesis

Petter Nilsson[1] · **Necmiye Ozay**[1] · **Jun Liu**[2]

Received: 19 February 2016 / Accepted: 9 March 2017 / Published online: 31 March 2017

Abstract This work is motivated by the problem of synthesizing switching protocols for continuous switched systems described by differential or difference equations, in a way that guarantees that the resulting closed-loop trajectories satisfy certain high-level specifications expressed in linear temporal logic. We introduce *augmented* finite transition systems as an abstract representation of the continuous dynamics; the augmentation consists in encodings of liveness properties that can be used to enforce progress in accordance with the underlying continuous dynamics. Abstraction and refinement relations that induce a preorder on this class of finite transition systems are established, and, by construction, this preorder respects the feasibility (i.e., realizability) of the synthesis problem. Hence, existence of a discrete strategy for one of these abstract finite transition systems guarantees the existence of a switching protocol for the continuous system that enforces the specification for all resulting trajectories. We show how abstractions and refinements can be computed for different classes of continuous systems through an incremental synthesis procedure that starts with a coarse abstraction and gradually refines it according to the established preorder relations. Finally, the incremental synthesis procedure is tailored to a class of temporal logic formulas by utilizing specific fixed point structures to enable localized updates in the refinement steps. The procedure is not guaranteed to terminate in general but we illustrate its practical applicability on numerical examples.

✉ Petter Nilsson
pettni@umich.edu

Necmiye Ozay
necmiye@umich.edu

Jun Liu
j.liu@uwaterloo.ca

[1] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

[2] Department of Applied Mathematics, University of Waterloo, Waterloo, ON, Canada

Keywords Correct-by-construction · Control synthesis · Abstraction refinement

1 Introduction

Motivated by the need for designing controllers to satisfy complex requirements, there has been significant recent interest in abstraction-based control synthesis techniques (Tabuada 2009). These techniques provide a unified way to handle state and input constraints, and possibly hybrid dynamics, which are common in many safety-critical systems. In particular, several techniques have been proposed to construct abstractions for switched systems with discrete-valued actuators, or, equivalently, systems whose operation is restricted to a finite set of predefined modes (Cámara et al. 2011; Yordanov et al. 2012; Gol et al. 2012; Liu et al. 2013; Ozay et al. 2013). These abstractions are used for synthesizing switching protocols that orchestrate the low-level dynamics in a way that ensures that the trajectories of the system satisfy high-level specifications, typically given in terms of temporal logic formulas.

By an abstraction, also known as a symbolic model (Tabuada 2009), we essentially mean a finite graph structure that captures some of the properties of the underlying continuous dynamics. If a given switched system satisfies certain stability conditions, it is possible to obtain a finite transition system that (approximately) bisimulates it. For instance, Girard et al. show that under certain stability conditions, and if the sampling time is chosen carefully, it is possible to find a finite transition system that approximately bisimulates the time-sampled trajectories of a continuous-time switched system (Girard et al. 2010). Gol et al. present results for constructing bisimilar finite abstractions for stable discrete-time switched systems (Gol et al. 2012). If the switched system does not satisfy the stability conditions, or if a finite bisimilar model with a good approximation quality requires too many states, it is still possible to construct finite transition systems that simulate the underlying dynamics or over-approximate it as defined in (Liu et al. 2013). By construction, a control strategy for such an abstract transition system can be implemented as a switching protocol for the concrete switched system in a way that preserves specification satisfaction.

Both simulation relations and over-approximation relations are based on the ability of a finite transition system to encompass and mimic the transitions of the concrete continuous system. Transitions are short-horizon properties, therefore it is difficult to capture long-horizon properties of the underlying dynamics with the usual transition system structures. This shortcoming manifests itself, for instance, through spurious cycles in the finite transition system, which may cause liveness properties to be unenforceable on the abstract transition system, despite being enforceable on the concrete continuous system.

In this paper we address this limitation by introducing *augmented* finite transition systems (AFTS), which generalize finite transition systems with additional liveness conditions that encode transience properties of the underlying dynamics. Appropriate preorder relations among these transition systems are established to facilitate abstraction-refinement based incremental control synthesis. We also present algorithms to compute this type of abstraction and its refinements for various continuous-time and discrete-time dynamical systems. In the second part of the paper, we focus on controller synthesis and propose an incremental synthesis algorithm based on an abstraction-refinement loop. The algorithm is specialized to AFTSs and a fragment of Linear Temporal Logic (LTL). In particular, by utilizing the specific fixed-point structures that appear in solving the synthesis problems in

this fragment, the runtime of the incremental synthesis algorithm is improved by enabling localized updates during abstraction refinement.

Ideas similar to AFTSs have previously been considered for verification (Kesten and Pnueli 2000; Batt et al. 2008). In the context of abstraction-based control synthesis, eliminating potentially spurious self-loops—which are special instances of spurious cycles—has been proposed as a post-processing step during synthesis (Yordanov et al. 2012; Coogan and Arcak 2015). AFTSs and associated synthesis algorithms provide a systematic methodology to describe all potential spurious cycles in a unified way, thus eliminating the need for post-processing. AFTS-based abstractions are also related to the notion of fair simulation relations studied for purely discrete systems (Baier and Katoen 2008). Incremental synthesis for partition refinement has been proposed for stochastic linear systems (Svorenova et al. 2015) and for deterministic abstractions (Mattila et al. 2015). Compared to this paper, both of these works consider a different fragment of LTL, namely GR(1), and the class of continuous dynamics is limited.

The preliminary conference version of this work appeared in (Nilsson and Ozay 2014). Some related results have also appeared in (Ozay et al. 2013) and (Sun et al. 2014). The current paper unifies these preliminary results, significantly extends the class of systems and specifications, and provides complete implementation details. We have chosen to organize the exposition in the following way. First, we introduce some preliminary set notation in Section 2, and define the grammar and semantics of LTL. Subsequently, we define switched systems in Section 3, together with the problem we seek to solve. In Section 4 we introduce AFTSs together with related ordering notions and show in Section 5 how AFTSs can be used as an abstraction of a switched system. The proposed abstraction-synthesis-refinement framework is presented in Section 6 and in the following Section 7 we restrict attention to a fragment of LTL and give efficient synthesis algorithms. We describe how a controller can be extracted from the synthesis algorithms in Section 8 and comment on elimination of Zeno behavior. Finally, we present illustrative numerical examples in Section 9 before the paper is concluded in Section 10.

2 Preliminaries

2.1 Notation

We introduce the following notation pertaining to sets. For sets X and Y, $X \subset Y$ indicates that X is a (not necessarily strict) subset of Y. We write set closure as $\mathrm{cl}(X)$, set interior as $\mathrm{int}(X)$, and set complement as X^C. For a function $f : X \to Y$ we define the image of a set $V \subset X$ as $f(V) := \{f(x) : x \in V\} \subset Y$. Conversely, we define the pre-image of a set $W \subset Y$ as $f^{-1}(W) := \{x \in X : f(x) \in W\}$. Finally, the power set, or set of all subsets, of X is denoted $2^X := \{Y : Y \subset X\}$.

2.2 Linear temporal logic

Linear Temporal Logic (LTL) is a formalism for specifying temporal properties. LTL has two types of operators: logical connectives and temporal modal operators. The logic connectives are those used in propositional logic: *negation* ($\neg$), *disjunction* ($\vee$), *conjunction* ($\wedge$) and *material implication* ($\implies$). The temporal modal operators include *next* ($\bigcirc$), *always* ($\Box$), *eventually* ($\Diamond$) and *until* ($\mathbf{U}$). An LTL formula over a finite set of atomic propositions (AP) can be defined inductively as follows:

(i) $\texttt{True}$ is an LTL formula,
(ii) any atomic proposition $p \in AP$ is an LTL formula, and
(iii) given LTL formulas φ and ψ, $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$, and $\varphi \mathbf{U} \psi$ are also LTL formulas.

The remaining derived operators are defined as follows: (i) $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, (ii) $\varphi \implies \psi := \neg\varphi \vee \psi$, (iii) $\Diamond\varphi := \texttt{True}\ \mathbf{U}\ \varphi$, and (iv) $\Box\varphi := \neg\Diamond\neg\varphi$.

Semantics of LTL LTL formulas are interpreted over *ω-words*, which are infinite sequences in 2^{AP}. Given an ω-word $w = w(0)w(1)w(2)\ldots$ in 2^{AP} and an LTL formula φ, we write $(w, i) \models \varphi$ if and only if w satisfies φ at a position $i \geq 0$, which is defined inductively as follows:

(i) For an atomic proposition $p \in AP$, $(w, i) \models p$ iff $p \in w(i)$;
(ii) $(w, i) \models \neg\varphi$ iff $(w, i) \not\models \varphi$;
(iii) $(w, i) \models \varphi \vee \psi$ iff $(w, i) \models \varphi$ or $(w, i) \models \psi$;
(iv) $(w, i) \models \bigcirc\varphi$ iff $(w, i + 1) \models \varphi$; and
(v) $(w, i) \models \varphi \mathbf{U} \psi$ iff there exists $j \geq i$ such that $(w, j) \models \psi$ and $(w, k) \models \varphi$ for all $k \in [i, j)$.

An ω-word w satisfies φ, written as $w \models \varphi$, if and only if $(w, 0) \models \varphi$. Given an arbitrary set Y and an observation map $h : Y \to 2^{AP}$, we can interpret LTL formulas for both discrete-time sequences and continuous-time functions taking values in Y. Let $y = y(0)y(1)y(2)\ldots$ be an infinite sequence in Y; then the ω-word generated by y is $h(y) := h(y(0))h(y(1))h(y(2))\ldots$. We say that y *satisfies* φ, written as $y \models \varphi$, if $h(y) \models \varphi$. For a continuous-time function $y : [0, \infty) \to Y$, an ω-word $w = w(0)w(1)w(2)\ldots$ in 2^{AP} is said to be consistent with the observation of y if and only if there exists a strictly increasing sequence $\{t_k\}_{k=0}^{\infty}$ in $[0, \infty)$ with $t_0 = 0$ and $t_k \to \infty$ as $k \to \infty$, such that $h(y(t)) = w(k)$ for all $t \in [t_k, t_{k+1})$. Note that for any such sequence $\{t_k\}_{k=0}^{\infty}$, all ω-words consistent with the observation of y are stutter equivalent (Baier and Katoen 2008). We say y satisfies φ, written as $y \models \varphi$, if there exists an ω-word w consistent with the observation of y such that $w \models \varphi$.

3 Problem setup

In this paper we consider control synthesis for switched dynamical systems to satisfy given linear temporal logic specifications. We use a cohesive notation that encompasses switched dynamical systems in both continuous and discrete time. In the following, a switched system is a tuple $\mathcal{S} = (X, \mathcal{U}, \{f_u\}_{u\in\mathcal{U}}, \mathcal{D}, AP, h_X)$, where

- $X \subset \mathbb{R}^n$ is a compact domain,
- $\mathcal{U}$ is a finite set that enumerates the modes,
- f_u is a mapping $X \to \mathbb{R}^n$ for all $u \in \mathcal{U}$,
- $\mathcal{D} \subset \mathbb{R}^d$ is a compact disturbance set,
- AP is a finite set of atomic propositions, and
- $h_X : X \to 2^{AP}$ is a state labeling function.

The evolution of the state $x \in \mathbb{R}^n$ of $\mathcal{S}$ is governed by

$$D^+x(t) = f_{\sigma(t)}(x(t), \delta(t)), \tag{1}$$

where

$$D^+x(t) := \begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}x(t), & \text{for continuous time,} \\ x(t+1), & \text{for discrete time.} \end{cases}$$

In Eq. 1, $\sigma(t) \in \mathcal{U}$ determines the active dynamical mode at time t, and $\delta(t) \in \mathcal{D}$ is a disturbance signal. In the continuous time case we assume that regularity conditions on the functions $\{f_u\}_{u\in\mathcal{U}}$, δ, and σ are fulfilled so that a weak[1] solution to (1) exists and is unique. It is enough to require that for all $u \in \mathcal{U}$, $f_u(x, \delta)$ Lipschitz continuous in $X \times \mathcal{D}$, and, in addition, that δ is continuous and that σ is piecewise constant, see e.g. (Walter and Thompson 1998, p. 121).

In the following we will refer to such weak solutions as *trajectories*, which will be denoted $\mathbf{x} : I \to X$ for a time interval I. In the continuous time case, $I = [0, \tau]$ or $I = [0, \infty)$, whereas in discrete time $I = \{0, 1, \ldots, T\}$ or $I = \mathbb{N}$. For the unbounded cases $I = [0, \infty)$ and $I = \mathbb{N}$, the corresponding trajectory is called *maximal*.

A *switching protocol* for a switched system $\mathcal{S}$ is a partial function $\pi : M \times X \to \mathcal{U}$ together with an internal state update function π_{int}. The function π maps an internal memory state $m \in M$ and system state $x \in X$ to an action $u \in U$, and the memory $m(t)$ follows some internal memory dynamics described by $\pi_{int}(m(t), x(t))$. The control synthesis problem considered in this paper can now be stated as follows.

Problem 1 *Given a switched system model $\mathcal{S} = (X, \mathcal{U}, \{f_u\}_{u\in\mathcal{U}}, \mathcal{D}, AP, h_X)$ in continuous or discrete time, a set of initial conditions X_0, and an LTL formula φ over AP, find a switching protocol π such that all resulting closed-loop trajectories starting in X_0 are maximal and satisfy φ.*

4 Augmented finite transition systems

We now introduce augmented finite transition systems and define relations between augmented finite transition systems and switched systems.

Definition 1 An **augmented finite transition system** (AFTS) is a tuple $\mathcal{T} = (Q, \mathcal{U}, \to_{\mathcal{T}}, \mathcal{G}, AP, h_Q)$, where

- Q is a finite set of states,
- $\mathcal{U}$ is a finite set of actions (control inputs),
- $\to_{\mathcal{T}} \subseteq Q \times \mathcal{U} \times Q$ is a transition relation,
- $\mathcal{G} : 2^{\mathcal{U}} \to 2^{2^Q}$ is a progress group map,
- AP is a finite set of atomic propositions, and
- $h_Q : Q \to 2^{AP}$ is a labeling function.

The difference between an AFTS and usual finite transition system definitions (e.g., (Baier and Katoen 2008)) is the progress group map. When there are no progress groups, i.e. $\mathcal{G}(U) = \emptyset$ for all $U \in 2^{\mathcal{U}}$, the AFTS is equivalent to a finite transition system. Given an AFTS $\mathcal{T}$, we denote by $\Pi(\mathcal{T})$ the finite transition system *induced by* $\mathcal{T}$, which is the finite transition system obtained from $\mathcal{T}$ by setting $\mathcal{G}(U) = \emptyset$ for all $U \in 2^{\mathcal{U}}$.

[1] A weak (Carathéodory) solution is absolutely continuous and satisfies (1) for almost all t in the Lebesque measure sense.

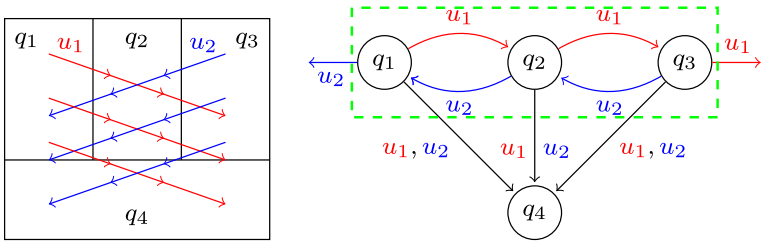

Fig. 1 Illustration showing how a multi-action progress group can encode information that restricts the behavior of an AFTS in a way that is useful for control purposes. The nondeterministic finite transition system to the right can be thought of as an abstraction (see Def. 5) of the switched system with two modes to the left. Without progress groups and starting in q_1, there is no way to choose between actions u_1 and u_2 to guarantee that the system will end up in q_4 since $q_1q_2q_3q_2q_1q_2\ldots$ is a valid trajectory. However, if $P := \{q_1, q_2, q_3\}$ is a progress group under actions $\{u_1, u_2\}$, i.e. $P \in G(\{u_1, u_2\})$, the state can not remain indefinitely in P while these actions are used. This information can be inferred from the switched system. Therefore, by always selecting u_1 in q_1 and u_2 in q_3, the state can be controlled to q_4 in finite time. This type of progress group information is exploited in the synthesis algorithms presented later in the paper. For this example, single-mode progress groups are not capable of encoding the same information

The progress group map $\mathcal{G}$ maps a subset of actions $U \in 2^{\mathcal{U}}$ to a set of subsets of states. A set $G \in \mathcal{G}(U)$ is called a *progress group under the action set* U and restricts the behavior of $\mathcal{T}$ in the following way: the system cannot remain indefinitely within G by exclusively choosing actions from U. In previous work (Sun et al. 2014; Nilsson and Ozay 2014), only single-action progress group maps of the form $\mathcal{G} : \mathcal{U} \to 2^{2^Q}$ were utilized. In this work, we allow for the extended notion of *multi-action* progress groups that allow a richer class of transience properties to be encoded. Figure 1 shows an example of how multi-action progress groups can encode synthesis-critical information that is not possible to encode with single-action progress groups.

Definition 2 An AFTS is said to be **well-formed** if the following holds for all $U \in 2^{\mathcal{U}}$ and for all subsets $V \in 2^G$ of all $G \in \mathcal{G}(U)$: there is a state $q_1 \in V$ such that for all actions $u \in U$ there exists a transition $(q_1, u, q_2) \in \to_{\mathcal{T}}$ such that $q_2 \notin V$.

This condition is required to comply with the notion of a progress group: if an AFTS is not well-formed it is possible to select actions so that the state remains indefinitely in some $V \subset G$. Another way to interpret well-formedness is that a progress group $G \in \mathcal{G}(U)$ can not contain a controlled invariant set for actions restricted to U.

We assume, without loss of generality, that all actions are enabled at every state.[2] That is, for all $q_1 \in Q$ and for all $u \in \mathcal{U}$, there exists at least one $q_2 \in Q$ such that $(q_1, u, q_2) \in \to_{\mathcal{T}}$. An *execution* ρ of an AFTS $\mathcal{T}$ is an infinite sequence of pairs $\rho = (q(0), u(0))\,(q(1), u(1))\,(q(2), u(2))\ldots$, where $(q(k), u(k), q(k+1)) \in \to_{\mathcal{T}}$ for all $k \geq 0$, that satisfies the progress conditions encoded by $\mathcal{G}$. That is,

$$\forall U \in 2^{\mathcal{U}},\ \forall G \in \mathcal{G}(U),\ \forall K \geq 0,\ \exists k > K \ \text{ s.t. } \ (q(k), u(k)) \notin G \times U. \tag{2}$$

[2] A dummy state q_d can be added, together with transitions from all forbidden state-action pairs, to obtain an AFTS that is equivalent for synthesis purposes.

Furthermore, the *word* produced by an execution ρ is an infinite sequence $w = h_Q(q(0))$ $h_Q(q(1))h_Q(q(2))\ldots$. A *control strategy* for an AFTS $\mathcal{T}$ is a partial function $\mu : (Q \times \mathcal{U})^* \times Q \to 2^{\mathcal{U}}$ that maps the execution history $\rho \in (Q \times \mathcal{U})^*$ and current state $q \in Q$ to a set of actions. A *μ-controlled execution* of $\mathcal{T}$ is an execution where for each step $k \geq 0$, the action $u(k)$ is chosen according to the control strategy μ.

Synthesizing a control strategy for a finite transition system in order to guarantee the satisfaction of an LTL formula by all controlled executions of the system is a well-understood problem (Bloem et al. 2012; Piterman and Pnueli 2006; Church 1962; Pnueli and Rosner 1989). Moreover, it is known that if there is a control strategy that guarantees the satisfaction of the specification, there is always a finite memory strategy (Grädel et al. 2002). That is, one can always choose $\mu : M \times Q \to 2^{\mathcal{U}}$ with internal update rule $m(k+1) = \mu_{int}(m(k), q(k))$ for $m(k) \in M$, where M is a finite set. Next, we show that controller synthesis for AFTSs with LTL specifications can be reduced to a controller synthesis problem for a finite transition system.

Proposition 1 *Given a well-formed AFTS $\mathcal{T} = (Q, \mathcal{U}, \to_{\mathcal{T}}, \mathcal{G}, AP, h_Q)$ and an LTL formula φ, let*

$$\varphi_{G,U} := \neg\Diamond\Box(q \in G \wedge u \in U) = \Box\Diamond\left((q \notin G) \vee (u \notin U)\right), \tag{3}$$

for a progress group G under an action set U. Then, there exists a control strategy for $\mathcal{T}$ that guarantees the satisfaction of φ, if and only if there exists a control strategy for $\Pi(\mathcal{T})$ that guarantees the satisfaction of the following modified specification φ' over an extended set of atomic propositions, defined as

$$\varphi' := \left(\bigwedge_{U \in 2^{\mathcal{U}}} \bigwedge_{G \in \mathcal{G}(U)} \varphi_{G,U}\right) \to \varphi. \tag{4}$$

Proof The progress group condition (2) is captured by the LTL formula (3). Therefore, the progress information can equivalently be stated as part of the formula φ'. □

Finite synthesis problems like these can be solved for general LTL specifications using software such as TuLiP (Filippidis et al. 2016), a toolbox that provides interfaces for multiple synthesis tools. Although the synthesis problem can be computationally challenging for general LTL specifications, there are fragments of LTL such as GR(1) with more favorable computational complexity (Piterman et al. 2006). If $\varphi \in$ GR(1), then also $\varphi' \in$ GR(1), so the algorithmic complexity of the synthesis problem is not affected by the addition of progress group encodings.

The next definition gives a relation between AFTSs, which leads to a preorder.

Definition 3 Given two AFTSs $\hat{\mathcal{T}} = (\hat{Q}, \mathcal{U}, \to_{\hat{\mathcal{T}}}, \hat{\mathcal{G}}, AP, \hat{h}_{\hat{Q}})$ and $\mathcal{T} = (Q, \mathcal{U}, \to_{\mathcal{T}}, \mathcal{G}, AP, h_Q)$, $\mathcal{T}$ is said to be a **refinement** of $\hat{\mathcal{T}}$ (or, $\hat{\mathcal{T}}$ is an **abstract model** of $\mathcal{T}$), denoted by $\hat{\mathcal{T}} \preceq \mathcal{T}$, if there exists a **refinement function** $\beta : Q \to \hat{Q}$ such that the following conditions hold.

(i) For all $q \in Q$, $h_Q(q) = \hat{h}_{\hat{Q}}(\beta(q))$.
(ii) For all $(q_1, u, q_2) \in \to_{\mathcal{T}}$, $(\beta(q_1), u, \beta(q_2)) \in \to_{\hat{\mathcal{T}}}$.

(iii) For all $U \subset \mathcal{U}$, for all $\hat{G} \in \hat{\mathcal{G}}(U)$, there exists $G \in \mathcal{G}(U)$ such that for all $\hat{q} \in \hat{G}$, we have $\beta^{-1}(\{\hat{q}\}) \subseteq G$.

It is easy to see that $\succeq$ is a preorder relation as it is reflexive and transitive. The reflexive property can be shown by taking β to be the identity map, while transitiveness follows by noting that if $\mathcal{T}_1 \succeq \mathcal{T}_2$ with the refinement function $\beta_{1,2}$, and $\mathcal{T}_2 \succeq \mathcal{T}_3$ with the refinement function $\beta_{2,3}$, then $\beta_{1,3} := \beta_{1,2} \circ \beta_{2,3}$ is a refinement function that verifies $\mathcal{T}_1 \succeq \mathcal{T}_3$.

Proposition 2 *Given $\hat{\mathcal{T}} \succeq \mathcal{T}$, the set of all words that can be generated by $\mathcal{T}$ is a subset of those that can be generated by $\hat{\mathcal{T}}$. Moreover, given an LTL formula φ over 2^{AP}, if it is realizable on $\hat{\mathcal{T}}$, then it is realizable on $\mathcal{T}$.*

Proof The set of generated words is restricted by the absence of transitions or the presence of progress groups. Conditions (i)-(ii) in Definition 3 imply that the states of $\hat{\mathcal{T}}$ are aggregations of disjoint sets of equally labeled states in $\mathcal{T}$. For every transition in the refinement $\mathcal{T}$ there is therefore a corresponding transition in the abstract model $\hat{\mathcal{T}}$, but $\hat{\mathcal{T}}$ may also have additional transitions (i.e., more non-determinism). Additionally, condition (iii) ensures that for each progress group $\hat{G}$ of the abstract model $\hat{\mathcal{T}}$, there is a corresponding progress group G of the refinement $\mathcal{T}$. Since $\mathcal{T}$ contains a subset of the transitions and a superset of the progress groups of $\hat{\mathcal{T}}$, all words generated by $\mathcal{T}$ can also be generated by $\hat{\mathcal{T}}$.

Given a control strategy $\hat{\mu} : M \times \hat{Q} \to \mathcal{U}$ with an internal update $\hat{\mu}_{int} : M \times \hat{Q} \to M$ for $\hat{\mathcal{T}}$ that induces trajectories with words satisfying φ, a control strategy $\mu : M \times Q \to \mathcal{U}$ for $\mathcal{T}$ with internal update $\mu_{int} : M \times Q \to M$ can be defined as

$$\mu(m, q) := \hat{\mu}(m, \beta(q)), \quad \mu_{int}(m, q) := \hat{\mu}_{int}(m, \beta(q)). \tag{5}$$

By the above, the set of words generated by μ will be a non-empty subset of those generated by $\hat{\mu}$, so φ is realizable also on $\mathcal{T}$. □

Next, we seek to establish a similar relation between switched systems and AFTSs. This necessitates the concept of *transience*—the continuous equivalent of progress groups.

Definition 4 Given a switched system $\mathcal{S} = (X, \mathcal{U}, \{f_u\}_{u \in \mathcal{U}}, \mathcal{D}, AP, h_X)$, a set $Y \subset X$ is **transient** on a set of modes $U \subset \mathcal{U}$ if and only if for any state $x_0 \in Y$, for any disturbance $\delta(t) \in \mathcal{D}$, and any $\sigma(t)$ taking values in U, there exists a bounded interval I such that the trajectory $\mathbf{x} : I \to X$ of $D^+x(t) = f_{\sigma(t)}(x(t), \delta(t))$ with $\mathbf{x}(0) = x_0$ leaves Y, i.e., $\mathbf{x}(I) \setminus Y \neq \emptyset$.

As with progress groups, there is an equivalent interpretation of transience in terms of controlled invariance: a set Y is transient on a set of modes $U \subset \mathcal{U}$ if and only if it does not contain a controlled invariant set when the mode signal is restricted to U and δ is regarded as a control input.

Definition 5 An AFTS $\mathcal{T} = (Q, \mathcal{U}, \to_{\mathcal{T}}, \mathcal{G}, AP, h_Q)$, is an **over-approximation** of the switched system $\mathcal{S} = (X, \mathcal{U}, \{f_u\}_{u \in \mathcal{U}}, \mathcal{D}, AP, h_X)$, denoted $\mathcal{T} \succeq \mathcal{S}$, if there exists an **abstraction function** $\alpha : X \to Q$, such that the following statements hold.

(i) For all $x \in X$, $h_X(x) = h_Q(\alpha(x))$.

(ii) Given states $q_1,\ q_2 \in Q$, if there exist $x_0 \in \alpha^{-1}(q_1)$, a compact time interval I (not a singleton), and some disturbance $\delta : I \to \mathcal{D}$ such that the corresponding trajectory $\mathbf{x} : I \to X$ of $D^+x(t) = f_u(x(t), \delta(t))$, with $\mathbf{x}(0) = x_0$, satisfies

$$\mathbf{x}\left(\max_{t\in I} t\right) \in \alpha^{-1}(q_2), \quad \mathbf{x}(I) \subset \alpha^{-1}(q_1) \cup \alpha^{-1}(q_2),$$

then $(q_1, u, q_2) \in \to_{\mathcal{T}}$.

(iii) The progress group map $\mathcal{G}$ is such that for all action sets $U \in 2^{\mathcal{U}}$, for all $G \in \mathcal{G}(U)$, the set $\alpha^{-1}(G)$ is transient on the mode set U of $\mathcal{S}$.

As shown next, the over-approximation as defined above preserves LTL realizability, which implies that a correct control strategy for an over-approximation $\mathcal{T} \succeq \mathcal{S}$ can be implemented as a switching protocol on $\mathcal{S}$.

Theorem 1 *Let $\mathcal{S}$ be a switched system that is over-approximated by an AFTS $\mathcal{T}$, i.e., $\mathcal{T} \succeq \mathcal{S}$. Furthermore, let φ be an LTL (LTL$\backslash\bigcirc$ for continuous time, since the next operator ($\bigcirc$) is not well defined) formula over the atomic propositions of $\mathcal{S}$ and $\mathcal{T}$. If there exists a control strategy for $\mathcal{T}$ that guarantees the satisfaction of φ, then there is a switching protocol for $\mathcal{S}$ that guarantees the satisfaction of φ.*

Proof We show that for any single-valued[3] control strategy $\mu : M \times Q \to \mathcal{U}$ for $\mathcal{T}$ with internal update rule $\mu_{int} : M \times Q \to M$, there is a corresponding switching protocol π for $\mathcal{S}$ with internal dynamics π_{int} such that each trajectory of $\mathcal{S}$ resulting from π corresponds to a μ-controlled execution in $\mathcal{T}$.

The switching protocol π is a function of its internal state $m(t)$. For continuous time, it is constructed inductively as follows for a trajectory $\mathbf{x}$ starting in x_0. First, given an internal state $m(t)$ at time t, the switching protocol outputs $\pi(m(t), \mathbf{x}(t)) := \mu(m(t), \alpha(\mathbf{x}(t))$. Secondly, let $q(0) := \alpha(x_0)$, $m_0 := \mu_{int}(\emptyset, q(0))$, and $t_1 := \inf\{t \geq 0 : \alpha(\mathbf{x}(t)) \neq q(0)\}$. Then the internal state of π on the interval $[0, t_1)$ is $m(t) := m_0$. Lastly, we define the update rule π_{int} of the internal state. For a time $t \geq t_k$, let the internal state of π be $m(t) = m_k$. We let $m(t)$ be constant on the interval $[t_k, t_{k+1})$, for $t_{k+1} := \inf\{t \geq t_k : \alpha(\mathbf{x}(t)) \neq \alpha(\mathbf{x}(t_k))\}$. We define $q(k+1) := \alpha(\lim_{\tau \searrow t_{k+1}} \mathbf{x}(\tau))$, where $\searrow$ indicates the limit from above, and let $m(t) := \mu_{int}(m_k, q(k+1))$ for times $t \geq t_{k+1}$.

This construction results in a strictly increasing[4] sequence $\{t_k\}$ in $[0, \infty)$ with $t_0 = 0$, such that the output maps produce the same word,

$$\begin{aligned} \pi(m(t), \mathbf{x}(t)) &= \mu(m_k, q_k) =: u(k), \\ h_X(\mathbf{x}(t)) &= h_Q(\alpha(\mathbf{x}(t))) = h_Q(q(k)), \end{aligned} \qquad \forall t \in [t_k, t_{k+1}). \tag{6}$$

For discrete time, the construction simplifies to initializing $q(0) := \alpha(x_0)$ and $m(0) := \mu_{int}(\emptyset, q(0))$. The switching protocol outputs $u(t) := \pi(m(t), \mathbf{x}(t)) := \mu(m(t), \alpha(\mathbf{x}(t)))$ and the internal state updates as $m(t+1) := \mu_{int}(m(t), q(t+1))$.

Also this construction leads to output equivalence between $\mathcal{T}$ and $\mathcal{S}$, given as

$$\pi(t) = u(t), \quad h_X(\mathbf{x}(t)) = h_Q(\alpha(\mathbf{x}(t))) = h_Q(q(t)), \quad \forall t = 0, 1, \ldots \tag{7}$$

[3] Taking values in $\mathcal{U}$, as opposed to in $2^{\mathcal{U}}$. Any set-valued control strategy can be restricted to a single-valued strategy by fixing a selection rule.

[4] In the continuous time case, some mild conditions on continuous implementations of the strategy being non-Zeno are necessary. See Section 8 for more details.

By construction, in both cases $q := q(0)q(1)q(2)\ldots$ is a μ-controlled execution of $\mathcal{T}$. If it satisfies φ, it follows from Eq. 6 (resp. (7)) that also $\mathbf{x}$ satisfies φ. □

5 Abstractions of switched systems

We now detail how an abstraction of a switched system

$$\mathcal{S} = (X, \mathcal{U}, \{f_u\}_{u\in\mathcal{U}}, \mathcal{D}, AP, h_X)$$

can be constructed as an AFTS. The abstraction will satisfy the properties of an over-approximation in Definition 5, which allows us to reason about trajectories of $\mathcal{S}$ by analyzing the abstraction algorithmically.

We consider abstractions based on a partition of the domain X. A partition can be defined in terms of the equivalence classes $X/\sim_\alpha$ of an abstraction function α and its associated equivalence relation $\sim_\alpha$:

$$x \sim_\alpha y \quad \text{iff} \quad \alpha(x) = \alpha(y).$$

In the following we assume that α takes a finite number of values, which makes also the corresponding partition $X/\sim_\alpha$ finite, and that each partition cell $\alpha^{-1}(q) = \{x : \alpha(x) = q\}$ satisfies the regularity property $\text{cl}\left(\text{int}\left(\alpha^{-1}(q)\right)\right) = \text{cl}\left(\alpha^{-1}(q)\right) \neq \emptyset$. From a computational point of view, the type of partitions that we use (e.g., consisting of hyper boxes or convex polyhedra with non-empty interior) naturally satisfy this assumption. We restrict attention to abstraction functions α with the property of being *proposition preserving* with respect to the set of atomic propositions AP. Being proposition preserving means that continuous states that are in the same equivalence class have identical truth evaluations, i.e., for $x, y \in X$, $x \sim_\alpha y \implies h_X(x) = h_X(y)$.

Consider now an AFTS $\mathcal{T}_\alpha = \{Q, \mathcal{U}, \rightarrow_\alpha, \mathcal{G}, AP, h_\alpha\}$, where $Q = \alpha(X)$, $h_\alpha := h_X \circ \alpha^{-1}$, and the construction of $\rightarrow_\alpha$ and $\mathcal{G}$ are described below.

Determine transitions $\rightarrow_\alpha$ We require a transition (q_1, u, q_2) to be added to the AFTS whenever there exists a corresponding trajectory between $\alpha^{-1}(q_1)$ and $\alpha^{-1}(q_2)$ in $\mathcal{S}$. What we are actually interested in is to limit spurious behaviors of the abstraction by obtaining certificates for the absence of such trajectories. This task is described in Algorithm 1, which systematically determines whether transitions exist between pairs of cells in the partition. It will typically produce a nondeterministic AFTS, i.e., for a given state-action pair (q_1, u) there may be several successors q_2 such that $(q_1, u, q_2) \in \rightarrow_{\mathcal{T}}$. Algorithm 1 depends on the following subroutine:

R.1 `isBlocked`$(C_1, C_2, f, \mathcal{D})$: Given two sets $C_1, C_2 \subset \mathbb{R}^n$, single-mode dynamics described by f, and a disturbance set $\mathcal{D}$, return `False` if there exists a non-singleton compact interval I and a trajectory $\mathbf{x} : I \to X$ of $D^+x(t) = f(x(t), \delta(t))$ that satisfies

$$\mathbf{x}(0) \in C_1, \quad \mathbf{x}\left(\max_{t\in I} t\right) \in C_2, \quad \mathbf{x}(I) \subset C_1 \cup C_2,$$

and `True` otherwise.

Algorithm 1 Compute transitions $\rightarrow_\alpha$

```
function ComputeTrans({f_u}_{u∈U}, D, Q, α)
    initialize →_α = ∅
    for u ∈ U do
      for (q_i, q_j) ∈ Q × Q do
        if not isBlocked(α^{-1}(q_i), α^{-1}(q_j), f_u, D) then
          add (q_i, u, q_j) to →_α
    return →_α
```

Remark 1 In the interest of keeping notation simple, we have omitted handling potential out-of-domain trajectories. If there is a trajectory $\mathbf{x}$ on an interval I of $D^+x(t) = f_u(x(t), \delta(t))$ such that

$$\mathbf{x}(I) \subset \alpha^{-1}(q) \cup X^C, \quad X^C \cap \mathbf{x}(I) \neq \emptyset,$$

then the action u must be avoided at state q. This can be encoded in the AFTS by adding a dummy state q_{out} to Q, and adding transitions (q, u, q_{out}) to $\rightarrow_\alpha$ for all state-action pairs (q, u) with potential out-of-domain trajectories, and then avoid q_{out} in synthesis by adding the specification $\square\neg q_{out}$. If $\texttt{isBlocked}(C, X^C, f_u, \mathcal{D}) = \texttt{True}$, then there are no out-of-domain trajectories from C corresponding to f_u.

Determine progress groups $\mathcal{G}$ We use Algorithm 2 to extract a progress group map from a switched system $\mathcal{S}$. In essence, it considers collections of discrete states and adds them to the progress group map if their pre-image under α is transient, as determined by the following subroutine:

R.2 $\texttt{isTransient}(C_1, \{f_u\}_{u\in U}, \mathcal{D})$: Given a set C_1, switched dynamics described by $\{f_u\}_{u\in U}$, and a disturbance set $\mathcal{D}$, return `False` if there exists a maximal trajectory $\mathbf{x}$ of $D^+x(t) = f_{\sigma(t)}(x(t), \delta(t))$, with mode switching restricted to σ taking values in U, such that $\mathbf{x}(I) \subset C_1$, and `True` otherwise.

Algorithm 1 will by default add many transitions of the form (q, u, q) to $\rightarrow_\mathcal{T}$. For instance, in continuous time $\texttt{isBlocked}(C, C, f_u, D) = \texttt{True}$ by definition. To eliminate spurious self-loops, single-action progress groups can be added for states q for which $\alpha^{-1}(\{q\})$ is transient. The same progress properties can however also be encoded in larger progress groups, as discussed later in the section.

Algorithm 2 Compute progress group $\mathcal{G}$

```
function ComputeProgGroup({f_u}_{u∈U}, D, Q, α)
    initialize G : 2^U → 2^{2^Q}
    for U ∈ 2^U do
      initialize G(U) = ∅,
      for V ∈ 2^Q do
        if isTransient(α^{-1}(V), {f_u}_{u∈U}, D) then
          add V to G(U)
    return G
```

Whether the subroutines `isBlocked` and `isTransient` can be implemented, and how efficiently that can be done, depends on the form of the dynamics (1) (e.g., linear, polynomial) and the type of sets induced by α (e.g., hyper boxes, polyhedra, semi-algebraic sets). In the end of this section we give convex optimization formulations that can be used to implement `isBlocked` and `isTransient` for various combinations of dynamics and set descriptions.

Algorithms 1 and 2 can typically be implemented more efficiently than the pseudo-code given here. For instance, at line 4 in Algorithm 1 it is sufficient to consider neighboring pairs of cells in continuous time, as transitions between all other pairs are blocked by definition. The search can also be heavily restricted in discrete time by calculating a global bound on transition distance and automatically infer `isBlocked` to be `True` for pairs of cells separated by a longer distance. Such localized search procedures improve the complexity of the algorithm.

Similarly, the exponential complexity of Algorithm 2 is impractical in most applications but can often be improved in practice. There is no additional benefit of either 1) finding progress groups that are subsets of another progress group under the same action set, or 2) finding a single-action progress group that consists of states that are already part of a multi-action progress group containing the same action. In these cases, the transience properties are already captured by the "larger" progress groups. A reasonable choice is therefore a heuristic search strategy that primarily focuses on finding large multi-action progress groups. In the abstraction-refinement loop described in Section 6, the progress group computed for the initial coarse abstraction where $|Q|$ is small is mapped through refinement steps, while search for new progress groups is conducted in localized candidate regions only. It is also worth noting that for systems with globally asymptotically stable equilibrium points per different subsets of inputs, the set of all cells that do not contain the equilibrium form a large progress group and computation reduces to finding the cells containing equilibrium points.

Intuitively, `isBlocked` and `isTransient` look for certificates of blocked transitions, and transient subsets, so that spurious trajectories in the abstraction can be eliminated. However, it is not necessary that `isBlocked` and `isTransient` are exact in order for the conditions of Definition 5 to hold. It is enough that they do not return false positives, as captured in the following theorem. For the same reason, it is not necessary to find all possible progress groups, although more progress groups might increase the quality of the approximation.

Theorem 2 *Assume that an abstracting AFTS $\mathcal{T}$ is constructed from a switched system $\mathcal{S}$ as described above, using a proposition preserving abstraction function and implementations of* `isBlocked` *and* `isTransient` *that do not return false positives (i.e., if* `True` *is returned for a given query, the conditions for* `True` *in R.1 resp. R.2 are indeed satisfied). Then $\mathcal{T}$ is well-formed and $\mathcal{T} \succeq \mathcal{S}$.*

Proof Condition (i) of Definition 5 is satisfied by the definition of h_α. If `isBlocked` does not return false positives all transitions present in $\mathcal{S}$ will be added to $\mathcal{T}$ by Algorithm 1, which assures the satisfaction of (ii). Similarly, if `isTransient` does not return false positives no progress groups will be added by Algorithm 2 unless their pre-image is transient as required by (iii). Thus $\mathcal{T} \succeq \mathcal{S}$.

We show that $\mathcal{T}$ is well-formed. Assume for contradiction that Eq. 2 is not satisfied. Then there exists $G \in \mathcal{G}(U)$ such that for some $V \subset G$,

$$\forall v_1 \in V, \exists u \in U \text{ s.t. } (v_1, u, v_2) \in \rightarrow_{\mathcal{T}} \implies v_2 \in V. \tag{8}$$

By the above, $\alpha^{-1}(V)$ is transient on U, meaning that all trajectories of $\mathcal{S}$ under modes in U will eventually exit V. In particular, this holds for trajectories generated by selecting actions according to Eq. 8. But by the same reasoning as above, the discrete analogues of these trajectories must be present also in $\mathcal{T}$ which is a contradiction of Eq. 8. □

We now give concrete implementations of `isBlocked` and `isTransient` for a variety of situations. We start with general formulations and then specialize to convex, efficient implementations for the special cases of linear and polynomial systems, for both discrete and continuous time. We note that some variants of `isBlocked` for linear and multi-affine dynamics on rectangular sets or simplices have appeared in the literature (Habets et al. 2006; Belta and Habets 2006; Girard and Martin 2012).

5.1 General formulations

For general sets $C_1, C_2 \subset X$ and dynamics described by $f : X \times \mathcal{D} \to \mathbb{R}^n$, the following optimization formulations can be used to determine whether there is a transition from C_1 to C_2.

5.1.1 `isBlocked`

For continuous time, we infer blocking by a Nagumo-type condition (Walter and Thompson 1998, Chapter 10, XVI). Intuitively, if the vector field along a surface is always pointing "inwards", then no trajectories of that vector field can cross the surface to the "outside". Let f be a continuous function, and $\hat{N}_{C_1}(x)$ be the normal cone[5] of $\mathrm{cl}(C_1)$ at x. Then, if the optimal value of

$$\begin{cases} \max\limits_{x,\delta} \ \hat{n}(x) \cdot f(x,\delta), \\ \text{s.t.} \quad x \in \mathrm{cl}(C_1) \cap \mathrm{cl}(C_2), \\ \qquad \delta \in \mathcal{D}, \\ \qquad \hat{n}(x) \in \hat{N}_{C_1}(x), \end{cases} \tag{BLK-CT}$$

is smaller than or equal to 0, the vector field f points "inwards" towards C_1 everywhere on $\mathrm{cl}(C_1) \cap \mathrm{cl}(C_2)$, which implies that $\texttt{isBlocked}(C_1, C_2, f, \mathcal{D}) = \texttt{True}$.

For discrete time, if the following program is infeasible,

$$\begin{cases} \text{find} \ x \in C_1, \ \delta \in \mathcal{D}, \\ \text{s.t.} \quad f(x,\delta) \in C_2, \end{cases} \tag{BLK-DT}$$

then $\texttt{isBlocked}(C_1, C_2, f, \mathcal{D}) = \texttt{True}$.

5.1.2 `isTransient`

As remarked earlier in the paper, the concept of transience is equivalent to the non-existence of a controlled invariant set. Therefore, existence of a function satisfying Lyapunov-like conditions can be used to infer that a given set is transient. For a decay rate $\epsilon > 0$, we

[5] The normal cone of a general set can be defined as follows (Clarke et al. 1998). The tangent cone of a set C at x is the cone $\hat{T}_C(x) := \{u : \forall\{x_i\} \to x, \ \forall\{t_i\} \to 0 \text{ from above}, \ \exists\{u_i\} \to u \text{ s.t.} x_i + t_i u_i \in C \text{ for all} i\}$. Informally, $\hat{T}_C(x)$ consists of directions u in which infinitesimal moves from x remain in C. The normal cone of C at x is then the dual of $\hat{T}_C(x)$: $\hat{N}_C(x) := \{v : u^T v \leq 0 \ \forall u \in \hat{T}_C(x)\}$.

propose the following search problem to determine transience of a bounded set C_1 under dynamics governed by $\{f_u\}_{u\in U}$:

$$\begin{cases} \text{find } B : \mathrm{cl}(C_1) \to \mathbb{R}, \\ \text{s.t. } \Delta_u B(x,\delta) \le -\epsilon, \quad \forall x \in \mathrm{cl}(C_1), \quad \forall \delta \in \mathcal{D}, \quad \forall u \in U, \end{cases} \tag{TRS}$$

where

$$\Delta_u B(x,\delta) := \begin{cases} \nabla B(x) \cdot f_u(x,\delta), & \text{for continuous time,} \\ B(f_u(x,\delta)) - B(x), & \text{for discrete time.} \end{cases}$$

Proposition 3 *If a differentiable function B satisfying (TRS) can be found, then* `isTransient`$(C_1, \{f_u\}_{u\in U}, \mathcal{D}) =$ `True`.

Proof Assume that a differentiable function B satisfying (TRS) exists. By compactness of $\mathrm{cl}(C_1)$ and continuity of B, B attains a minimal value on $\mathrm{cl}(C_1)$. Assume for contradiction that there is an unbounded interval I and a maximal trajectory $\mathbf{x} : I \to \mathrm{cl}(C_1)$ generated by a disturbance $\delta : I \to \mathcal{D}$ and a switching signal $\sigma : I \to U$.

For continuous time,

$$-\epsilon \ge \nabla B(\mathbf{x}(t)) \cdot f_{\sigma(t)}(\mathbf{x}(t), \delta(t)) = \frac{\mathrm{d}}{\mathrm{d}t} B(\mathbf{x}(t)), \quad \text{for almost all } t \in I,$$

Integrating up to time T yields $-\epsilon T \ge B(\mathbf{x}(T)) - B(\mathbf{x}(0))$, which contradicts the boundedness of B on $\mathrm{cl}(C_1)$.

For discrete time,

$$-\epsilon \ge B(f_{\sigma(t)}(\mathbf{x}(t), \delta(t)) - B(\mathbf{x}(t)) = B(\mathbf{x}(t+1)) - B(\mathbf{x}(t)).$$

Summing from $t = 0$ to T gives

$$-\epsilon(T+1) \ge \sum_{t=0}^{T} (B(\mathbf{x}(t+1)) - B(\mathbf{x}(t))) = B(\mathbf{x}(T+1)) - B(\mathbf{x}(0)),$$

due to a telescopic sum. This again contradicts the boundedness of B on $\mathrm{cl}(C_1)$. □

Existence of such a function B can also be shown to be a necessary condition for transience of compact sets in both continuous (Lin et al. 1996) and discrete (Jiang and Wang 2002) time.

5.2 Linear system, polyhedra

When f is affine and the sets $\mathrm{cl}(C_1)$ and $\mathrm{cl}(C_2)$ are polyhedra, `isTransient` and `isBlocked` can be implemented as linear programs.

Assume that f is an affine mapping, i.e. $f(x,\delta) = Ax + E\delta + K$, and that the sets $\mathrm{cl}(C_i) = \{x : H_i x \le h_i\}$ for $i = 1, 2$, and $\mathcal{D} = \{\delta : H_{\mathcal{D}}\delta \le h_{\mathcal{D}}\}$ are (convex) polyhedra. The intersection $\mathrm{cl}(C_1) \cap \mathrm{cl}(C_2)$ is then the common facet of $\mathrm{cl}(C_1)$ and $\mathrm{cl}(C_2)$ which is itself a polyhedron with a normal $\hat{n}$ (assuming it has co-dimension 1).

Then, (BLK-CT) simplifies to the linear program

$$\begin{cases} \max\limits_{x,\delta} & \hat{n}^T(Ax + E\delta + K), \\ \text{s.t} & H_1 x \le h_1, \ H_2 x \le h_2, \ H_{\mathcal{D}}\delta \le h_{\mathcal{D}}. \end{cases} \tag{BLK-CT-LIN}$$

For intersections with co-dimension higher than 1, the normal of any separating hyperplane between C_1 and C_2 can be used to obtain a blocking certificate.

Linear programs attain their optimal values at vertices. Thus, in order to find the optimal value of Eq. BLK-CT-LIN it is enough to evaluate the objective function at the combinations of all vertices of the intersection $\mathrm{cl}(C_1) \cap \mathrm{cl}(C_2)$ with all vertices of $\mathcal{D}$. If the vertex enumeration is computationally cheap, as it is for instance for hyper boxes, the enumeration method can be beneficial compared to solving (BLK-CT-LIN) with a standard LP solver.

Similarly, (BLK-DT) becomes the linear feasibility problem

$$\begin{cases} \text{find} & x, \delta, \\ \text{s.t} & H_1 x \leq h_1,\ H_{\mathcal{D}}\delta \leq h_{\mathcal{D}}, \\ & H_2(Ax + E\delta + K) \leq h_2, \end{cases} \tag{BLK-DT-LIN}$$

which is feasible if and only if $\texttt{isBlocked}(C_1, C_2, f, \mathcal{D}) = \texttt{False}$.

For linear systems, convex sets are controlled invariant if and only if they contain a controlled fixed point (in discrete time this is a special case of Kakutani's fixed point theorem, see (Feuer and Heymann 1976) for continuous time), so transience can be determined by proving the absence of such fixed points. We therefore propose the following linear program in the *single-mode* case, since a single-mode switched linear system is just a linear system. For the multi-mode case it is preferable to use the polynomial approach described in Section 5.3 below. For continuous time, we get

$$\begin{cases} \text{find} & x,\ \delta, \\ \text{s.t.} & H_1 x \leq h_1,\ H_{\mathcal{D}}\delta \leq h_{\mathcal{D}}, \\ & A_u x + E_u \delta + K_u = 0, \end{cases} \tag{TRS-CT-LIN}$$

and for discrete time,

$$\begin{cases} \text{find} & x,\ \delta, \\ \text{s.t.} & H_1 x \leq h_1,\ H_{\mathcal{D}}\delta \leq h_{\mathcal{D}}, \\ & A_u x + E_u \delta + K_u = x. \end{cases} \tag{TRS-DT-LIN}$$

In these cases, $\texttt{isTransient}(C_1, \{f_u\}, \mathcal{D}) = \texttt{True}$ if and only if Eq. TRS-CT-LIN (resp. (TRS-DT-LIN)) is infeasible.

5.3 Polynomial system, semi-algebraic sets

Assume instead that f is polynomial and that $\mathrm{cl}(C_1), \mathrm{cl}(C_2)$, and $\mathcal{D}$ are simple semi-algebraic sets described by $\mathrm{cl}(C_j) = \{x : g^i_{C_j}(x) \geq 0,\ i = 1, \ldots, k_{C_j}\}$ with $\deg(g^i_{C_j}) \leq d_{C_j}$ for $j = 1, 2$, and $\mathcal{D} = \{d : g^i_{\mathcal{D}}(d) \geq 0,\ i = 1, \ldots, k_{\mathcal{D}}\}$ with $\deg(g^i_{\mathcal{D}}) \leq d_{\mathcal{D}}$.

We first introduce some notation required to state polynomial optimization problems. Let $\mathbb{R}_d[x_1, \ldots, x_n]$ be the set of real polynomials over the variables $\{x_i\}_{i=1}^n$ and of degree at most d. Similarly, we use the notation $\Sigma_d[x_1, \ldots, x_n] \subset \mathbb{R}_d[x_1, \ldots, x_n]$ for the set of non-negative polynomials of degree at most d. To enable tractable optimization over the space of non-negative polynomials, the optimization must be restricted to convex subsets of $\Sigma_d[x_1, \ldots, x_n]$. For instance, it can be restricted to the set of sums-of-squares polynomials $\Sigma_d^{SOS}[x_1, \ldots, x_n]$, or to the set of scaled-diagonally-dominant sums-of-squares polynomials $\Sigma_d^{SDSOS}[x_1, \ldots, x_n]$. Positivity constraints translate to semi-definite constraints in the SOS case (Parrilo 2003) and to second-order cone programs in the SDSOS case (Ahmadi and Majumdar 2014); software such as YALMIP (Löfberg 2004) and SPOTless[6] automate

[6]SPOTless handles SDSOS optimization through an add-on by Anirudha Majumdar available at https://github.com/spot-toolbox/spotless.

the translation. For a more thorough introduction to optimization in the space of positive polynomials we refer to the tutorial paper (Ahmadi and Parrilo 2014) and references therein.

The optimization formulations of `isBlocked` and `isTransient` below are conservative due to truncation of the maximal polynomial degree. Crucially, this conservatism does not result in false positives (cf. Proposition 4).

For continuous time, an upper bound of the optimal value of Eq. BLK-CT can be found by fixing a bound $2d$ on maximal polynomial degree, and solving

$$\begin{cases} \min\limits_{\alpha,\sigma^i_{C_j},\sigma^i_{\mathcal{D}}} & \alpha, \\ \text{s.t} & \alpha - \hat{n}(x)\cdot f(x,\delta) - \sum_{i=1}^{k_{C_1}} \sigma^i_{C_1}(x,\delta) g^i_{C_1}(x) \\ & \quad - \sum_{i=1}^{k_{C_2}} \sigma^i_{C_2}(x,\delta) g^i_{C_2}(x) \\ & \quad - \sum_{i=1}^{k_{\mathcal{D}}} \sigma^i_{\mathcal{D}}(x,\delta) g^i_{\mathcal{D}}(\delta) \in \Sigma_{2d}[x,\delta], \\ & \sigma^i_{C_j}(x,\delta) \in \Sigma_{2d-d_{C_j}}[x,\delta],\ i=1,\ldots,k_{C_j},\ j=1,2, \\ & \sigma^i_{\mathcal{D}}(x,\delta) \in \Sigma_{2d-d_{\mathcal{D}}}[x,\delta],\ i=1,\ldots,k_{\mathcal{D}}. \end{cases} \tag{BLK-CT-POL}$$

The normal vector $\hat{n}(x)$ is chosen as the gradient of the level set function $g^{i^*}_{C_1}$ that induces the boundary between C_1 and C_2, $\hat{n}(x) := \pm\nabla G^{i^*}_{C_1}(x)$ with a sign that implies outward direction from C_1. In the degenerate cases where i^* is not unique, just as in the linear case, any $g^i_{C_1}(x)$ that separates C_1 from C_2 can be used to obtain a blocking certificate. The optimization is over the scalar α and the positive polynomial multipliers $\sigma^i_{C_j}$ and $\sigma^i_{\mathcal{D}}$. If the optimal value of Eq. BLK-CT-POL is less than 0 we can infer that $\texttt{isBlocked}(C_1, C_2, f, \mathcal{D}) = \texttt{True}$, since $\alpha \le 0$ implies that $0 \ge \hat{n}(x)\cdot f(x,\delta)$ for all $(x,\delta) \in (\text{cl}(C_1) \cap \text{cl}(C_2)) \times \mathcal{D}$.

For discrete time, we can state k_{C_2} optimization problems for $i = 1,\ldots,k_{C_2}$,

$$\begin{cases} \min\limits_{\alpha,\sigma^j_{C_1},\sigma^j_{\mathcal{D}}} & \alpha \\ \text{s.t} & \alpha - g^i_{C_2}(f(x,\delta)) - \sum_{j=1}^{k_{C_1}} \sigma^j_{C_1}(x,\delta) g^j_{C_1}(x) \\ & \quad - \sum_{j=1}^{k_{\mathcal{D}}} \sigma^j_{\mathcal{D}}(x,\delta) g^j_{\mathcal{D}}(\delta) \in \Sigma_{2d}[x,\delta], \\ & \sigma^j_{C_1}(x,\delta) \in \Sigma_{2d-d_{C_1}}[x,\delta],\ j=1,\ldots,k_{C_1}, \\ & \sigma^j_{\mathcal{D}}(x,\delta) \in \Sigma_{2d-d_{\mathcal{D}}}[x,\delta]\ j=1,\ldots,k_{\mathcal{D}}. \end{cases} \tag{BLK-DT-POL-i}$$

If the optimal value of Eq. BLK-DT-POL-i for any i is less than 0, that implies $g^i_{C_2}(f(x,\delta)) < 0$ for all $x \in C_1$ and all $\delta \in \mathcal{D}$. Thus Eq. BLK-DT is clearly infeasible so $\texttt{isBlocked}(C_1, C_2, f, \mathcal{D}) = \texttt{True}$. This procedure essentially amounts to over-approximating the one step reachable set of C_1 and intersecting it with C_2. If the intersection is empty no transitions are possible.

Finally, `isTransient` can be implemented using the following polynomial equivalent of Eq. TRS:

$$\begin{cases} \text{find} & B \in \mathbb{R}_{2d}[x],\ \sigma^{i,u}_{C_1},\ \sigma^{i,u}_{\mathcal{D}} \\ \text{s.t.} & -\Delta_u B(x,\delta) - \epsilon - \sum_{i=1}^{k_{C_1}} \sigma^{i,u}_{C_1}(x,\delta) g^i_{C_1}(x) \\ & \quad - \sum_{i=1}^{k_{\mathcal{D}}} \sigma^{i,u}_{\mathcal{D}}(x,\delta) g^i_{\mathcal{D}}(\delta) \in \Sigma_{2d}[x,\delta],\ \ \forall u \in U, \\ & \sigma^{i,u}_{C_1}(x,\delta) \in \Sigma_{2d-d_{C_1}}[x,\delta],\ \ i=1,\ldots,k_{C_1},\ \ \forall u \in U, \\ & \sigma^{i,u}_{\mathcal{D}}(x,\delta) \in \Sigma_{2d-d_{\mathcal{D}}}[x,\delta],\ \ i=1,\ldots,k_{\mathcal{D}},\ \ \forall u \in U. \end{cases} \tag{TRS-POL}$$

If a B satisfying these conditions can be found, then $-\Delta_u B(x,\delta) - \epsilon \geq 0$ on $C_1 \times \mathcal{D}$ uniformly over $u \in U$, thus $\texttt{isTransient}(C_1, \{f_u\}_{u\in U}, \mathcal{D}) = \texttt{True}$.

6 Abstraction-synthesis-refinement

In order to reduce the computational effort associated with computing a fine-grained uniform partition, we propose an algorithm that starts with a coarse partition and iteratively refines it in "promising" areas of the state space. Once an initial coarse abstraction $\mathcal{T} = (Q, \mathcal{U}, \rightarrow_{\mathcal{T}}, \mathcal{G}, AP, h_Q)$ of a switched system $\mathcal{S}$ has been constructed, the synthesis algorithm attempts to compute a winning set $W \subset Q$ in which the specification φ can be enforced. In addition, it also computes a losing set $L \subset Q$, starting from where there is no hope that φ can be satisfied even after further refinement. The refinement procedure, which produces increasingly tight over-approximations of $\mathcal{S}$, is iterated until the synthesis algorithm can produce a control strategy that solves Problem 1 (i.e., the initial set is covered by the winning set: $X_0 \subset \alpha^{-1}(W)$), until a counterexample is obtained that proves the unrealizability of the specification (i.e., the losing set intersects the initial set: $\alpha^{-1}(L) \cap X_0 \neq \emptyset$), or until the computational resources are exhausted. The algorithm, illustrated in Fig. 2, contains three main components **Abstraction**, **Synthesis**, and **Refinement**.

We are interested in making the winning and losing sets as large as possible in order to extract as much information as possible about the realizability of φ. To this end we also compute *candidate* winning and losing sets C_W and C_L that serve as feedback to the refinement step. These sets represent areas where cell refinement may allow the actual winning and losing sets to be expanded in a later synthesis stage. In the following subsections we describe the **Abstraction**, **Synthesis**, and **Refinement** procedures in detail.

6.1 Abstraction

To construct an initial abstraction of $\mathcal{S} = (X, \mathcal{U}, \{f_u\}_{u\in\mathcal{U}}, \mathcal{D}, AP, h_X)$, any abstraction function α on X that is proposition preserving with respect to the set of atomic propositions AP will suffice. Given such an abstraction function α—a natural choice is the coarsest α that is proposition preserving—an abstracting AFTS $\mathcal{T}$ can be constructed following the procedure in Section 5.

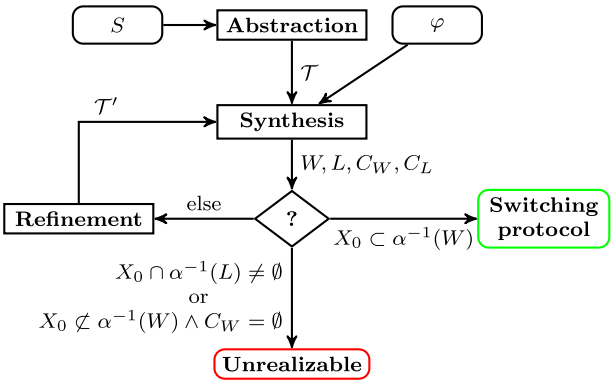

Fig. 2 Schematic view of the abstraction-synthesis-refinement algorithm

6.2 Synthesis

In the synthesis step, winning, losing, and candidate winning and losing sets are computed. In order to define these sets, we introduce four operators $\mathrm{Win}_{\sharp_1,\sharp_2}(\mathcal{T},\varphi)$ that take an AFTS $\mathcal{T}$ and an LTL formula φ and return the set of states in Q where φ can be enforced. In $\mathrm{Win}_{\sharp_1,\sharp_2}$, the symbols $\sharp_1$ and $\sharp_2$ are both quantifiers in the set $\{\exists,\forall\}$—thus making the number of possible combinations four—that denote whether the actions ($\sharp_1$), and the nondeterminism ($\sharp_2$), are controllable ($\exists$), or not ($\forall$). For instance, $\mathrm{Win}_{\exists,\exists}(\mathcal{T},\varphi)$ returns the set of initial conditions from where φ can be enforced, provided that both the actions and the nondeterminism in $\mathcal{T}$ are controllable. We remark that such finite synthesis problems over an AFTS can be solved algorithmically for general LTL specifications (cf. Proposition 1).

Given the Win operators, we can introduce the concepts of winning, losing, and candidate winning and losing sets as follows:

$$W := \mathrm{Win}_{\exists,\forall}(\mathcal{T},\varphi), \quad \text{(winning)}, \tag{9a}$$

$$C_W := \mathrm{Win}_{\exists,\exists}(\mathcal{T},\varphi) \setminus W, \quad \text{(candidate winning)}, \tag{9b}$$

$$L := \mathrm{Win}_{\forall,\forall}(\mathcal{T},\neg\varphi), \quad \text{(losing)}, \tag{9c}$$

$$C_L := \mathrm{Win}_{\forall,\exists}(\mathcal{T},\neg\varphi) \setminus L, \quad \text{(candidate losing)}. \tag{9d}$$

As can be seen, W and C_W are defined using $(\exists,\sharp)$ quantifiers, which corresponds to actions being controllable. The difference between the two is that nondeterminism is assumed to be uncontrollable for W, but controllable for C_W. The sets L and C_L are defined in an equivalent manner but with uncontrollable actions.

Given an AFTS $\mathcal{T}$ together with a specification φ, the synthesis step amounts to computing (9a)–(9d). These computations can be performed for general formulas using LTL synthesis algorithms, however, for structured specifications more efficient algorithms may be devised. In addition to efficiency, a second benefit of using formula-tailored algorithms is that the computational effort for consecutive synthesis steps can be reduced by using previous synthesis results, which we call *incremental synthesis*. Thirdly, in the definitions above it can be shown that $(W \cup L)^C = C_W = C_L$[7], which implies that every discrete state that is not winning or losing is both a candidate winning and a candidate losing state. Restricting attention to specifications with specific structure makes it possible to compute localized candidate sets, a modification that potentially enhances the efficiency of the refinement procedure. In Section 7 we provide tailored algorithms for a meaningful fragment of LTL that enable localized refinement, as well as efficient incremental synthesis.

6.3 Refinement

In the event that an acceptable winning set was not found, i.e., $X_0 \not\subset \alpha^{-1}(W)$, and that infeasibility of Problem 1 was not proven, we propose to refine the abstracting AFTS in the hope that a tighter over-approximation will reveal more information about the satisfiability

[7]Follows by noting that $W^C = \mathrm{Win}_{\forall,\exists}(\mathcal{T},\neg\varphi)$ and $L^C = \mathrm{Win}_{\exists,\exists}(\mathcal{T},\varphi)$, which implies $(W \cup L)^C = \mathrm{Win}_{\exists,\exists}(\mathcal{T},\varphi) \cap \mathrm{Win}_{\forall,\exists}(\mathcal{T},\neg\varphi)$.

of φ on $\mathcal{S}$. We focus the refinement on regions of the state space where it may be beneficial, i.e., the candidate winning and losing sets.

Specifically, if at step t of the abstraction-refinement loop the current AFTS is $\mathcal{T}^t = (Q^t, \mathcal{U}, \rightarrow_{\mathcal{T}^t}, \mathcal{G}^t, AP, h^t_Q)$ with abstraction function α^t, we select for $q_i \in C_W \cup C_L$ a cell $\alpha_t^{-1}(q_i) \subset X$ and split it into two parts. This results in two new discrete states q_{i_1} and q_{i_2} and we can define $Q^{t+1} = (Q \setminus \{q_i\}) \cup \{q_{i_1}, q_{i_2}\}$ and a new abstraction function $\alpha^{t+1} : X \rightarrow Q^{t+1}$ as $\alpha^{t+1}(x) := \alpha^t(x)$ for $x \notin (\alpha^t)^{-1}(q_i)$, and such that the pair $\left((\alpha^{t+1})^{-1}(q_{i_1}), (\alpha^{t+1})^{-1}(q_{i_2})\right)$ forms a 2-cell partition of $(\alpha^t)^{-1}(q_i)$. We can define a corresponding refinement function $\beta : \mathcal{Q}^{t+1} \rightarrow \mathcal{Q}^t$ as $\beta := \alpha^t \circ (\alpha^{t+1})^{-1}$.

In order to update the transitions and progress groups for the new transition system, Algorithms 1 and 2 could be used to determine $\rightarrow_{\mathcal{T}^{t+1}}$ and $\mathcal{G}^{t+1}$. However, since the refinement only locally modifies the partition structure, the vast majority of all transitions and progress groups remain intact. Thus, we only need to re-compute transitions involving the cells $(\alpha^{t+1})^{-1}(q_{i_1})$ and $(\alpha^{t+1})^{-1}(q_{i_2})$, which reduces the required computational effort drastically. For progress groups, if $G \in \mathcal{G}^t(U)$, then $\beta^{-1}(G)$ can be added to $\mathcal{G}^{t+1}(U)$ since the pre-image remains unchanged: $(\alpha^t)^{-1}(G) = (\alpha^{t+1})^{-1}\left(\beta^{-1}(G)\right)$. However, additional progress groups that satisfy (iii) of Definition 5 may have been revealed by the refinement, so a search for new progress groups may be appropriate. By construction, the following holds.

Proposition 4 *The refined AFTS* $\mathcal{T}^{t+1} = (Q^{t+1}, \mathcal{U}, \rightarrow_{\mathcal{T}^{t+1}}, \mathcal{G}^{t+1}, AP, h^t_Q \circ \beta)$ *satisfies*

$$\mathcal{T}^t \succeq \mathcal{T}^{t+1} \succeq \mathcal{S}.$$

There are in general many possibilities for selecting a cell for splitting, and the splitting itself can be done in several ways. Multiple cells can also be selected for splitting in the same refinement step. The next result shows that there are no "dead ends" in the space of all refinements, regardless of the refinement heuristics.

Proposition 5 *For any pair of abstracting AFTSs* $\mathcal{T}_1 \succeq \mathcal{S}$ *and* $\mathcal{T}_2 \succeq \mathcal{S}$*, there exists an AFTS* $\mathcal{T}_3 \succeq \mathcal{S}$ *such that* $\mathcal{T}_1 \succeq \mathcal{T}_3$ *and* $\mathcal{T}_2 \succeq \mathcal{T}_3$*. In other words, there exists a refinement* $\mathcal{T}_3$ *of* $\mathcal{T}_2$ *such that* $\mathcal{T}_3$ *approximates* $\mathcal{S}$ *at least as well as* $\mathcal{T}_1$ *does.*

Proof It is sufficient to "overlay" the partitions of $\mathcal{T}_1 = (Q_1, \mathcal{U}, \rightarrow_1, \mathcal{G}_1, AP, h_{Q_1})$ and $\mathcal{T}_2 = (Q_2, \mathcal{U}, \rightarrow_2, \mathcal{G}_2, AP, h_{Q_2})$ to create a new AFTS with the required property. Specifically, if α_1 and α_2 are abstraction functions for $\mathcal{T}_1$ and $\mathcal{T}_2$, let $\alpha_3 : X \rightarrow Q_1 \times Q_2$ be defined by $\alpha_3(x) = (\alpha_1(x), \alpha_2(x))$.

We construct $\rightarrow_3$ as follows:

- For each $(q_1, u, \bar{q}_1) \in \rightarrow_1$, add $((q_1, q_2), u, (\bar{q}_1, \bar{q}_2))$ to $\rightarrow_3$ for all q_2 such that $(q_1, q_2) \in \alpha_3(X)$ and all $\bar{q}_2$ such that $(\bar{q}_1, \bar{q}_2) \in \alpha_3(X)$.
- For each $(q_2, u, \bar{q}_2) \in \rightarrow_2$, add $((q_1, q_2), u, (\bar{q}_1, \bar{q}_2))$ to $\rightarrow_3$ for all q_1 such that $(q_1, q_2) \in \alpha_3(X)$ and all $\bar{q}_1$ such that $(\bar{q}_1, \bar{q}_2) \in \alpha_3(X)$.

Finally, adding $\alpha_3 \circ \alpha_1^{-1}(G_1)$ and $\alpha_3 \circ \alpha_2^{-1}(G_2)$ to $\mathcal{G}_3(U)$ for all $G_1 \in \mathcal{G}_1(U)$ and all $G_2 \in \mathcal{G}_2(U)$ produces the required object as $\mathcal{T}_3 := (\alpha_3(X), \mathcal{U}, \rightarrow_3, \mathcal{G}_3, AP, h_3)$, for $h_3(q_1, q_2) = h_1(q_1) = h_2(q_2)$ (well defined due to proposition preservation). $\mathcal{T}_3$ is a refinement of both $\mathcal{T}_1$ and $\mathcal{T}_2$ with refinement functions $\beta_i(q_1, q_2) := q_i$ for $i = 1, 2$. □

7 Structured synthesis

We now restrict attention to the LTL fragment consisting of specifications on the form

$$\varphi = \Box A \wedge \Diamond\Box B \wedge \left(\bigwedge_{i\in I} \Box\Diamond C^i\right), \tag{10}$$

for subsets $A, B, C^i \subset Q$ defining atomic propositions. This fragment encompasses specifications of invariance, persistence, and recurrence types, which together can be used to express a range of practical requirements. The same fragment was considered in (Wolff et al. 2013), where efficient synthesis algorithms were provided for FTSs. Here, we give algorithms that also take advantage of the information contained in the progress groups of an AFTS.

First we introduce notation which will allow us to express winning and losing sets of LTL formulas as fixed points of certain operators, under different choices of quantifiers as to what is controllable. The internals of the fixed point operators are then leveraged to define localized candidate winning and losing sets, and to enable incremental synthesis.

7.1 Primal fixed point operators

We define the following basic one-step reachability operators for an AFTS $\mathcal{T} = (Q, \mathcal{U}, \rightarrow_{\mathcal{T}}, \mathcal{G}, AP, h_Q)$, where as before $\sharp_1, \sharp_2 \in \{\exists, \forall\}$,

$$\mathrm{Pre}^{\mathcal{T},U}_{\sharp_1,\sharp_2}(V) = \{q_1 \in Q : \sharp_1(u \in U)\ \sharp_2\ (q_2 \ \text{ s.t } \ (q_1, u, q_2) \in \rightarrow_{\mathcal{T}}), \quad q_2 \in V\}. \tag{11}$$

For instance, $\mathrm{Pre}^{\mathcal{T},U}_{\exists,\forall}(V)$ is the set of all states that can be controlled to be in V at the next time step by selecting actions in U, regardless of how the nondeterminism in $\mathcal{T}$ is resolved. Similarly, $\mathrm{Pre}^{\mathcal{T},U}_{\forall,\forall}(V)$ is the set of states from where every transition ends in V—regardless of the action and how the nondeterminism is resolved. Using these operators as building blocks, we introduce algorithms that together are capable of computing the winning set of a specification of type (10). In the following $\sharp$ denotes either $\exists$ or $\forall$.

We first give an algorithm to compute the winning set of the fundamental specification $B\ \mathbf{U}\ Z$ as the smallest fixed point of an expanding iteration. Just as the operators $\Box$ and $\Diamond$ are defined in terms of ($\mathbf{U}$) (c.f. Section 2.2), we can compute the winning set of Eq. 10 by constructing higher-level fixed points that use the ($\mathbf{U}$) fixed point as a subroutine. The hierarchy of nested fixed points presented here is inspired by an algorithm from (Piterman and Pnueli 2006) that computes the winning set of a similar specification. We assume that the non-determinism is uncontrollable, i.e., we compute the winning sets $\mathrm{Win}^{\mathcal{T}}_{\exists,\forall}(B\ \mathbf{U}\ Z)$ and $\mathrm{Win}^{\mathcal{T}}_{\forall,\forall}(B\ \mathbf{U}\ Z)$. As shown in Appendix A, they are equal to the fixed point value X_∞ of the iteration scheme below:

$$\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(B\ \mathbf{U}\ Z) = \begin{cases} X_0 = \emptyset, \\ X_{k+1} = Z \cup \left(B \cap \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(X_k)\right) \cup \mathrm{PGPre}^{\mathcal{T}}_{\sharp,\forall}(X_k, B), \end{cases} \tag{12}$$

where the inner operator $\mathrm{PGPre}^{\mathcal{T}}_{\sharp,\forall}$ is defined as a union over fixed points Y_∞ of an iteration scheme.

$$\mathrm{PGPre}^{\mathcal{T}}_{\exists,\forall}(Z,B) := \bigcup_{U\in 2^{\mathcal{U}}} \bigcup_{G\in\mathcal{G}(U)} \mathrm{Inv}^{U,G}_{\exists}(Z,B), \tag{13a}$$

$$\mathrm{PGPre}^{\mathcal{T}}_{\forall,\forall}(Z,B) := \bigcup_{G\in\mathcal{G}(\mathcal{U})} \mathrm{Inv}^{\mathcal{U},G}_{\forall}(Z,B), \tag{13b}$$

$$\mathrm{Inv}^{U,G}_{\sharp}(Z,B) := \begin{cases} Y_0 = (G\cap B)\setminus Z, \\ Y_{k+1} = Y_k \cap \mathrm{Pre}^{\mathcal{T},U}_{\sharp,\forall}(Y_k\cup Z). \end{cases} \tag{13c}$$

The invariance-like operator $\mathrm{Inv}^{\mathcal{T}}_{\sharp,\forall}(Z,B)$ calculates a set Y_∞ contained in $G\cap B$ from where the state can either remain inside Y_∞ or enter Z. Since it can not remain inside $G\supset Y_\infty$ indefinitely due to the progress group condition, Z will eventually be reached. Defined as the union over all progress groups, the result $\mathrm{PGPre}^{\mathcal{T}}_{\sharp,\forall}(Z,B)$ comprises all points from where progress properties can be leveraged to enforce an eventual transition to Z, while remaining in B. In the $(\forall,\forall)$-case, only progress groups over all modes are taken into account; to leverage the progress property of G for $G\in\mathcal{G}(U)$ the actions must be constrained to U which is not possible if $U\neq\mathcal{U}$ and actions are uncontrollable.

We remark that the set sequence $\{X_k\}_{k\geq 1}$ generated by Eq. 12 is monotonically increasing with respect to inclusion. It will therefore converge to X_∞ in a finite number of steps since the state space is finite. Equivalent convergence properties hold for the other iterative algorithms presented in this section.

Based on the until operator, the winning set of a second specification that turns out to be useful can now be computed as the fixed point value W_∞ of the following iteration scheme:

$$\begin{aligned} &\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left((B\ \mathbf{U}\ Z)\vee\Box\left(B\wedge\left(\textstyle\bigwedge_{i\in I}\Diamond C^i\right)\right)\right) \\ &\quad= \begin{cases} W_0 = Q, \\ \tilde{Z}^i_{k+1} = Z\cup\left(B\cap C^i\cap\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(W_k)\right), \\ W_{k+1} = \bigcap_{i\in I}\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(B\ \mathbf{U}\ \tilde{Z}^i_{k+1}\right). \end{cases} \end{aligned} \tag{14}$$

The soundness and completeness of this algorithm are established in Appendix A. The case $I=\emptyset$—i.e. when the property has no recurrence part—is not well defined in Eq. 14 but can be handled by setting $I=\{1\}$ and $C^1=\texttt{True}$ which gives an equivalent specification.

Finally, the winning set of the specification of interest $\varphi=\Box A\wedge\Diamond\Box B\wedge\left(\bigwedge_{i\in I}\Box\Diamond C^i\right)$ can be computed by defining a fixed point on top of Eq. 14. To satisfy $\Box A$, we first compute $V_{inv}=\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box A)$ and restrict the synthesis to this controlled invariant set; the restriction amounts to modifying each call for computing $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(B\ \mathbf{U}\ Z)$ to instead compute

$\text{Win}^{\mathcal{T}}_{\sharp,\forall}((B \cap V_{inv}) \mathbf{U} (Z \cap V_{inv}))$.[8] Given the restriction, the winning set is obtained as the fixed point V_∞ of this iteration scheme:

$$\text{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi) = \begin{cases} V_{inv} = \text{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box A), \\ \text{Restrict synthesis to } V_{inv}, \\ V_0 = \emptyset, \\ \tilde{Z}_{k+1} = \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(V_k) \cup \text{PGPre}^{\mathcal{T}}_{\sharp,\forall}(V_k, Q), \\ V_{k+1} = \text{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\left(B \mathbf{U} \tilde{Z}_{k+1}\right) \vee \Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond C^i\right)\right)\right). \end{cases} \tag{15}$$

Again, a proof of soundness and completeness of this algorithm is provided in Appendix A.

7.2 Dual fixed point operators

In order to compute the losing set of φ, an algorithm for the winning set of $\neg\varphi$ is required. By duality, $\text{Win}^{\mathcal{T}}_{\exists,\forall}(\neg\varphi) = \text{Win}^{\mathcal{T}}_{\forall,\exists}(\varphi)^C$ and $\text{Win}^{\mathcal{T}}_{\forall,\forall}(\neg\varphi) = \text{Win}^{\mathcal{T}}_{\exists,\exists}(\varphi)^C$, but the algorithms above do not apply for controllable nondeterminism. The reason is that controllable nondeterminism conflicts with progress groups; certain sequences of nondeterminism resolutions violate the progress property, which invalidates the contracting algorithms above, in particular the $\text{Inv}^{U,G}$ operator. Instead, we introduce dual algorithms that compute the winning set of the negated specification. As before, the soundness and correctness of the three algorithms below are established in Appendix A.

The first algorithm computes the winning set of $\bigvee_{i\in I}\left[\left(B^i \mathbf{U} Z\right) \vee \Box B^i\right]$ as the union X_∞ of fixed points X^J_∞ for all nonempty subsets J of I. It can be seen as the dual of Eq. 12 since $\neg(\psi_1 \mathbf{U} \psi_2) = (\psi_1 \wedge \neg\psi_2) \mathbf{U} (\neg\psi_1 \wedge \neg\psi_2) \vee \Box(\psi_1 \wedge \neg\psi_2)$.

$$\begin{aligned} &\text{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\bigvee_{i\in I}\left[\left(B^i \mathbf{U} Z\right) \vee \Box B^i\right]\right) \\ &= \begin{cases} X^J_0 = Q \quad \text{for all } J \in 2^I \text{ s.t } J \neq \emptyset, \\ X^J_{k+1} = Z \cup \left(\left(\bigcap_{i\in J} B^i\right) \cap \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}\left(\bigcup_{K\in 2^J} X^K_k\right)\right), \\ X_\infty = \bigcup_{J\in 2^I} X^J_\infty. \end{cases} \end{aligned} \tag{16}$$

Secondly, we build upon (16) to obtain the following algorithm that returns the winning set of $\Diamond Z \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)$ as a fixed point W_∞:

$$\begin{aligned} &\text{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Diamond Z \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)\right) \\ &= \begin{cases} W_0 = \emptyset, \\ \tilde{Z}_{k+1} = \left(Z \cup \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(W_k) \cup \text{PGPre}^{\mathcal{T}}_{\sharp,\forall}(W_k, Q)\right), \\ W_{k+1} = \text{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\bigvee_{i\in I}\left[\left(B^i \mathbf{U} \tilde{Z}_{k+1}\right) \vee \Box B^i\right]\right). \end{cases} \end{aligned} \tag{17}$$

[8] As explained in the proof, an amended "always" specification $\Box A$ will be propagated through the fixed points down to the level of Eq. 12. Since the winning set of $\Box V^{inv} \wedge (B \mathbf{U} Z)$ is equal to the winning set of $\left((B \cap V^{inv}) \mathbf{U} (Z \cap V^{inv})\right)$ for a controlled invariant set V^{inv}, the restriction technique is correct. The propagation terms have been omitted to improve readability.

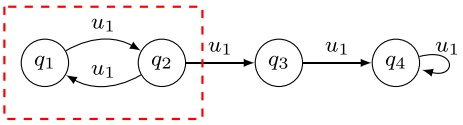

Fig. 3 For the AFTS depicted, if $\{q_1, q_2\} \in \mathcal{G}(u_1)$, then $\text{Win}^{\mathcal{T}}_{\exists,\forall}(\Diamond\{q_4\}) = \{q_1, q_2, q_3, q_4\}$. However, the winning set of the same specification without the progress group information is just $\text{Win}^{\Pi(\mathcal{T})}_{\exists,\forall}(\Diamond\{q_4\}) = \{q_3, q_4\}$, since $q_1q_2q_1q_2\ldots$ is a valid trajectory of $\Pi(\mathcal{T})$ that never reaches q_4

Finally, (17) is used as a subroutine in the computation of the winning set of $\Diamond A \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right) \vee \Box\Diamond C$—the dual of the specification of interest φ. The winning set is equal to the fixed point V_∞ of the following iteration scheme:

$$\begin{aligned}\text{Win}^{\mathcal{T}}_{\sharp,\forall}\ &\left(\Diamond A \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right) \vee \Box\Diamond C\right)\\ &= \begin{cases} V_0 = Q, \\ \tilde{Z}_{k+1} = A \cup \left(C \cap \text{Pre}^{\mathcal{T}}_{\sharp,\forall}(V_k)\right), \\ V_{k+1} = \text{Win}_{\sharp,\forall}\left(\Diamond\tilde{Z}_{k+1} \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)\right). \end{cases}\end{aligned} \tag{18}$$

By using duality, we can now compute winning sets for both φ and $\neg\varphi$ also in the $(\sharp, \exists)$ cases. We define $\bar{\sharp}$ as the quantifier dual to $\sharp$, that is, $\bar{\exists} := \forall$ and $\bar{\forall} := \exists$. Then,

$$\begin{aligned}&\text{Win}^{\mathcal{T}}_{\sharp,\exists}\left(\Box A \wedge \Diamond\Box B \wedge \bigwedge_{i\in I}\left(\Box\Diamond C^i\right)\right)\\ &\qquad = \text{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Diamond A^C \vee \left(\bigvee_{i\in I} \Diamond\Box (C^i)^C\right) \vee \Box\Diamond B^C\right)^C,\\ &\text{Win}^{\mathcal{T}}_{\sharp,\exists}\left(\Diamond A \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right) \vee \Box\Diamond C\right)\\ &\qquad = \text{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Box A^C \wedge \Diamond\Box C^C \wedge \left(\bigwedge_{i\in I} \Box\Diamond (B^i)^C\right)\right)^C.\end{aligned}$$

Remark 2 There is an interesting separation between winning sets that depend on progress group information, and those that do not. As illustrated in Fig. 3; $\text{Win}^{\mathcal{T}}_{\sharp,\forall}(\Diamond B) \neq \text{Win}^{\Pi(\mathcal{T})}_{\sharp,\forall}(\Diamond B)$ in general, but $\text{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box A) = \text{Win}^{\Pi(\mathcal{T})}_{\sharp,\forall}(\Box A)$.[9] The underlying intuition is that progress groups enforce liveness conditions; they therefore affect the winning set of a liveness specification $\Diamond B$, but not that of a safety specification $\Box A$. By duality, the converse holds in the $(\sharp, \exists)$-cases.

[9]The winning set of $\Box A$ can be computed for instance using the following special case of Eq. 14: $W_0 = Q$, $W_{k+1} = A \ \mathbf{U}\ \left(A \cap \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(W_k)\right)$. By induction it can be shown that when Eq. 12 is used to compute W_{k+1} it converges to $A \cap \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(W_k)$ after one iteration which implies that no progress group information was used.

7.3 Incremental synthesis

Equipped with the fixed point iterations from the previous subsections we now give algorithms to compute a winning set W and a losing set L for a specification of the form Eq. 10. These algorithms provide efficient means of computing (9a) and (9c), as opposed to solving a general synthesis problem with the progress groups encoded in LTL. By unwinding the fixed points on which the winning and losing sets are defined, we also obtain *localized* versions of the candidate winning and losing sets that are more specific than their general counterparts (9b) and (9d). Localized candidate sets are provided as feedback to guide the refinement step, as illustrated in Fig. 2.

Winning set (∃, ∀) The winning set W is defined in Eq. 14 as $W = \mathrm{Win}^{\mathcal{T}}_{\exists,\forall}(\varphi)$, so it can be computed using Eq. 15.

Losing set (∀, ∀) The losing set L is equal to $\mathrm{Win}^{\mathcal{T}}_{\forall,\forall}(\neg\varphi)$, which can be computed using Eq. 18.

Remark 3 Due to the subroutine (16), the computation of the losing set scales exponentially with the number of C^i's. This is expected, with $\neg\varphi$ being a disjunction of objectives. However, a smaller losing set does not affect correctness of the method, so if scalability is an issue the following conservative losing set can be used:

$$\mathrm{Win}^{\mathcal{T}}_{\forall,\forall}(\Diamond\neg A) \cup \left(\bigcup_{i\in I} \mathrm{Win}^{\mathcal{T}}_{\forall,\forall}\left(\Diamond\Box\neg C^i\right)\right) \cup \mathrm{Win}^{\mathcal{T}}_{\forall,\forall}(\Box\Diamond\neg B)\,.$$

Candidate winning set (∃, ∃) We next suggest a heuristic for a localized version of the candidate winning set, that is typically smaller than its general counterpart (9b). The winning set computation (15) is performed as a nested fixed point operation; any enlargement of the inner fixed points may potentially enlarge the final winning set as well. By systematically unwrapping the fixed point algorithms involved in Eq. 15, as presented in detail in Appendix B, we arrive at the following candidate winning set C_W:

$$\begin{aligned} C_W \;=\; & \left(\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\exists,\exists}(V_\infty) \setminus V_\infty\right) \cup (W_1 \setminus V_1) \\ & \cup \left(B \cap \left(\textstyle\bigcup_{j\in I} \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\exists,\exists}(W_{1,j}) \setminus W_{1,j}\right)\right) \cup (A \setminus V_{inv})\,. \end{aligned} \tag{19}$$

Here, V_1, V_∞, and V_{inv} are intermediate results from Eq. 15, and

$$W_1 = \bigcap_{j\in I} W_{1,j}, \quad W_{1,j} = \mathrm{Win}^{\mathcal{T}}_{\exists,\forall}\left(B \;\mathbf{U}\; \left(B \wedge C^i\right)\right).$$

Candidate losing set (∀, ∃). Finally, we also give a localized heuristic version of Eq. 9d as follows:

$$C_L \;=\; (V_1 \setminus V_\infty) \cup \left(\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\forall,\exists}(V_1) \setminus V_1\right) \cup \left(\textstyle\bigcup_i B^i \setminus W_1\right). \tag{20}$$

This is a set where the losing set may be possible to expand in a refined augmented transition system. Again, V_1 and V_∞ are intermediate results from Eq. 18, and

$$W_1 = \mathrm{Win}^{\mathcal{T}}_{\exists,\exists}\left(\bigvee_{i\in I}\left[\Box B^i \vee \left(B^i \;\mathbf{U}\; (A \cup C)\right)\right]\right).$$

A motivation behind this choice of C_L is provided in Appendix B.

Apart from being able to extract localized candidate sets, an advantage of the algorithms presented above is that certain winning set computations can be performed incrementally over refinement cycles (cf. Fig. 2). Smallest fixed points of an expanding algorithm always grow after refinement. Therefore, fixed points from previous iterations can be mapped to the refined partition and the algorithm can be initialized with these sets to speed up convergence to the smallest fixed point. To exemplify, let $V_\infty^t = \mathrm{Win}_{\exists,\forall}^{\mathcal{T}^t}\left(\Diamond\Box B \wedge \left(\bigwedge_{i\in I} \Box\Diamond C^i\right)\right)$. Then, if $\mathcal{T}^t \succeq \mathcal{T}^{t+1}$ with a refinement function β, we can compute $V_\infty^{t+1} = \mathrm{Win}_{\exists,\forall}^{\mathcal{T}^t}\left(\Diamond\Box B \wedge \left(\bigwedge_{i\in I} \Box\Diamond C^i\right)\right)$ using a modified version of Eq. 15:

$$V_\infty^{t+1} = \begin{cases} V_0 = \beta^{-1}(V_\infty^t), \\ V_{k+1} = \mathrm{Win}_{\exists,\forall}^{\mathcal{T}}\left(\left(B \;\mathbf{U}\; \mathrm{Pre}_{\exists,\forall}^{\mathcal{T},\mathcal{U}}(V_k)\right) \vee \Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond C^i\right)\right)\right). \end{cases}$$

In the same way, results for $\mathcal{T}^t$ of other expanding fixed point computations can be saved to get a "warm start" when computing the same smallest fixed points for $\mathcal{T}^{t+1}$. This does however not apply for contracting algorithms since the greatest fixed point for a contracting algorithm is smaller in $\mathcal{T}^t$ than in $\mathcal{T}^{t+1}$.

8 Controller extraction

For controller implementation, a control strategy μ that enforces the specification in the winning set W must be extracted. All fixed points encountered in Eq. 15 when computing W are ultimately defined in terms of $\mathrm{Pre}_{\exists,\forall}^{\mathcal{T},\mathcal{U}}$. It is therefore enough to extract control strategies during calls to this operator and map the results into higher-level fixed points together with memory variables that control a hierarchy of partial reachability objectives. In particular, this is done by extracting the actions that satisfy the existence condition $\exists u \in \mathcal{U}$ in Eq. 11. If the synthesis problem is solved with a general LTL solver, a control strategy can be extracted together with the winning set.

Remark 4 Certain algorithms in Section 7 are based on taking set complements. In these cases an enforcing strategy can not be extracted; only the winning set is obtained. However, primal algorithms are available for all necessary specifications in the $(\exists, \forall)$-case that is of interest when computing W.

As shown in Theorem 1, a control strategy μ for the AFTS $\mathcal{T} = (Q, \mathcal{U}, \rightarrow_{\mathcal{T}}, \mathcal{G}, AP, h_Q) \succeq \mathcal{S}$ can be implemented as a switching protocol on $\mathcal{S}$. There are in general several winning actions for any given state of μ. This control freedom can used to pursue performance objectives, or to eliminate potential continuous-time Zeno behavior, as described next.

Zeno behavior For continuous-time switched systems, it may happen that the solution found for $\mathcal{T}$ induces Zeno behavior when implemented on $\mathcal{S}$, as illustrated in Fig. 4. In addition to being undesirable due to its non-physical properties, Zeno behavior prohibits maximality of continuous-time trajectories, and thus prevents liveness-type specifications from being fulfilled. On the contrary, if Zeno behavior is not induced, the continuous switching signal will be piecewise constant which ensures maximality of trajectories as per the discussion in Section 3. Below we discuss how Zeno behavior can be eliminated. For

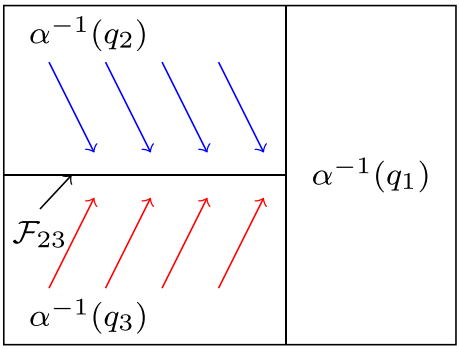

Fig. 4 Assume *red* and *blue* *arrows* correspond to vector fields for two different actions *red* and *blue*. Let the winning discrete control strategy require picking the *blue* action in q_2 and the *red* action in q_1. Implementing an abstraction-based switching protocol to reach $\alpha^{-1}(q_1)$ from $\alpha^{-1}(q_2) \cup \alpha^{-1}(q_3)$ might induce Zeno behavior at the facet $\mathcal{F}_{23}$

simplicity, we consider a memoryless[10] control strategy $\mu_0 : Q \to 2^{\mathcal{U}}$. It induces a closed loop graph $T_{\mu_0} = (Q, E_{\mu_0})$ with edge set E_{μ_0} given by

$$E_{\mu_0} = \bigcup_{q_1 \in Q} \{(q_1, q_2) : (q_1, u, q_2) \in \to_{\mathcal{T}} \text{ for some } u \in \mu_0(q_1)\}.$$

This graph describes (an over-approximation of) all discrete transitions that might occur at runtime. We say that a simple cycle $(q_1, q_2, \ldots, q_k)$ in T_{μ_0} is *Zeno-prone* if there exists a point that is a limit point of all corresponding partition cells, i.e. $\bigcap_{i=1}^{k} \mathrm{cl}\left(\alpha^{-1}(q_i)\right) \neq \emptyset$. All cycles of length 2 are automatically Zeno-prone. Since Zeno-prone cycles are by definition a local concept in the graph (the maximal length of a Zeno-prone cycle is the maximum number of jointly adjacent cells in the abstraction partition) it is significantly faster to enumerate all Zeno-prone cycles than to enumerate all simple cycles.

If there are no Zeno-prone cycles, Zeno behavior will not be present in $\mathcal{S}$. Specifically, if we can find a restricted controller $\bar{\mu}_0 : Q \to 2^{\mathcal{U}}$ such that $\bar{\mu}_0(q) \subset \mu_0(q)$ for all q, and such that its induced graph $T_{\bar{\mu}_0}$ is free from Zeno-prone cycles, implementing $\bar{\mu}_0 \circ \alpha$ as a feedback controller on the continuous time switched system will result in both correct and Zeno-free behavior.

We also propose a heuristic for eliminating Zeno behavior. In addition, it reduces the number of switches which may be beneficial in non-ideal systems where switches don't occur instantaneously. Given a control strategy $\mu : (Q \times \mathcal{U})^* \times Q \to 2^{\mathcal{U}}$, we define for $\rho(k) = (q(0), u(0)) \ldots (q(k-1), u(k-1))$ a control strategy $\tilde{\mu}$ as

$$\tilde{\mu}(\rho(k), q(k)) = \begin{cases} u(k-1), & \text{if } u(k-1) \in \mu(\rho(k), q(k)), \\ \text{any} u \in \mu(\rho(k), q(k)) & \text{otherwise.} \end{cases}$$

In essence, this control strategy uses the same action until it is necessary to select a different one in order to ensure future correctness. If Zeno behavior is still present when using $\tilde{\mu}$ on $\mathcal{S}$, further refinement of Zeno-prone areas may be necessary in order to eliminate Zeno behavior.

Finally, a third method to ensure Zeno-free trajectories is to time-discretize the continuous-time system and solve a discrete-time synthesis problem. When implementing the obtained control strategy as a continuous-time switching protocol, the sample time used

[10] For controllers with (finite) memory, the same analysis can be done using lifts and projections with respect to a space where states are augmented with the internal controller state.

in time-discretization can be used as the controller sampling rate to ensure a minimum dwell time for switching, and thus render Zeno behavior impossible. Under Lipschitz assumptions on the vector fields, margins can be added to (for safety)—or removed from (for liveness)—the atomic propositions to ensure that the final closed-loop system satisfies the specification in continuous time.

9 Examples

In this section we present two examples to demonstrate the effectiveness of the proposed approach. The first example compares the performance of finite transition systems to that of AFTSs, while the second example compares the proposed abstraction-refinement procedure to the approach in Sun et al. (2014) that uses non-incremental synthesis on a uniform state-space partition partition. The examples are solved with our prototype implementation that partitions the state space into hyper boxes. It is available at https://github.com/pettni/abstr-refinement.

9.1 Numerical example

We consider a continuous-time switched system, taken from (Ozay et al. 2013), with four modes $u \in \mathcal{U} \doteq \{1, 2, 3, 4\}$ and with the following dynamics:

$$f_1 = \begin{bmatrix} -x_2 - 1.5x_1 - 0.5x_1^3 \\ x_1 - x_2^2 + 2 + \delta \end{bmatrix}, \qquad f_2 = \begin{bmatrix} -x_2 - 1.5x_1 - 0.5x_1^3 \\ x_1 - x_2 + \delta \end{bmatrix},$$
$$f_3 = \begin{bmatrix} -x_2 - 1.5x_1 - 0.5x_1^3 + 2 \\ x_1 + 10 + \delta \end{bmatrix}, \quad f_4 = \begin{bmatrix} -x_2 - 1.5x_1 - 0.5x_1^3 - 1.5 \\ x_1 + 10 + \delta \end{bmatrix}, \quad (21)$$

where δ is the disturbance. The domain is taken to be $X = [-2, 2] \times [-1.5, 3]$ and we define the sets $X_a = [-2, -0.5] \times [-1.5, -1]$ and $X_b = [-1, -0.2] \times [0.5, 1.8]$ with associated atomic propositions $A := x \notin X_a$ and $B := x \in X_b$. The goal is to synthesize a switching protocol to guarantee the satisfaction of the specification $\varphi := \Box A \wedge \Diamond \Box B$.

We ran the proposed incremental abstraction-refinement algorithm using AFTSs to find a switching protocol together with a winning set from where this protocol is guaranteed to satisfy the specification. We considered a disturbance level of $|\delta| \leq 0.5$ as well as the case without disturbance ($|\delta| = 0$). As a comparison, we also ran the same algorithm using finite transition systems without progress groups, where we used the transience only to eliminate self-loops. Figure 5 shows the resulting winning sets after 80 iterations. The results clearly indicate that using AFTSs as abstractions reduces the conservatism. It is also worth mentioning that when using AFTSs, the algorithm terminated after 19 iterations in the case without disturbance, and after 70 iterations in the case with disturbance, since no candidate sets were left and the exact maximal winning set of the continuous problem was recovered.

For this example, the `isBlocked` and `isTransient` functions were computed based on an SDSOS-based implementation of the programs (BLK-CT-POL) and (TRS-POL).

9.2 Radiant systems in buildings

The second example is a hydronic radiant system for buildings in which chilled supply water is pumped through a concrete slab which in turn acts as a heat reservoir that moderates the

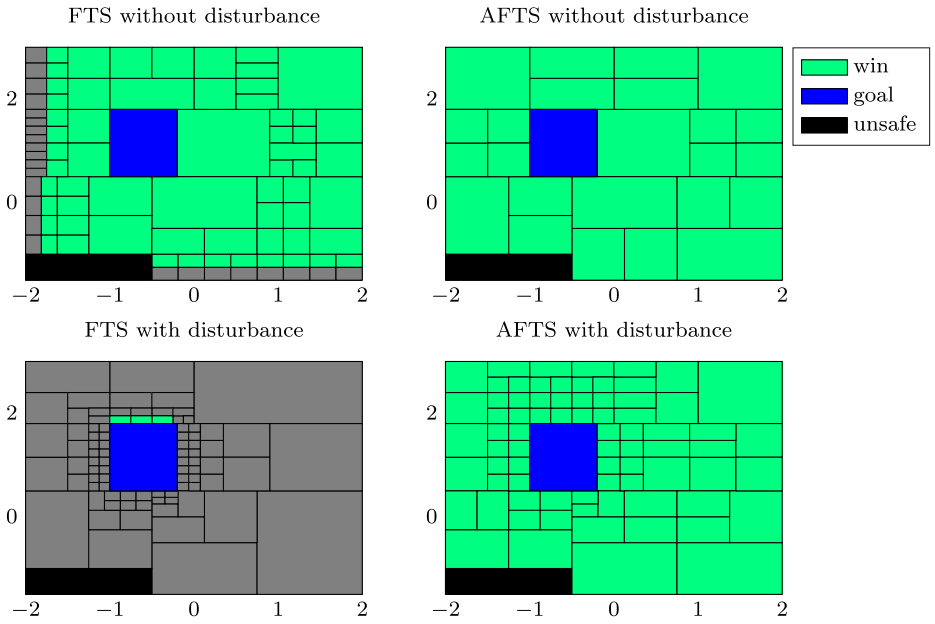

Fig. 5 Final partitioning of the domain of the numerical example for four different tests. Winning sets computed using the proposed incremental abstraction refinement framework are shown in green. For this example abstractions based on augmented finite transition systems outperform FTS-based abstractions by allowing a larger winning set to be computed using fewer discrete states. The incremental refinement process creates a finer partition in relevant areas of the state space

building temperature. By controlling the pump it is possible to regulate the temperature in the building. Such a system can be modeled as an RC circuit (Romaní et al. 2016)

$$C_i \dot{T}_i = \sum_{j \neq i} \frac{1}{R_{ij}} \left(T_j - T_i\right) + q_i, \tag{22}$$

where C_i is the thermal capacitance $[J/K]$ of zone i that is assumed to have homogeneous temperature T_i, $R_{ij} = R_{ji}$ is the thermal resistance $[K/W]$ of the barrier between zone i and zone j, and q_i is the heat added $[W]$ to zone i. The added heat may for example come from persons utilizing a room, or sunshine, which both vary over the course of a day. We want to describe a system with two rooms connected to the same hydronic system and opt for a model with five temperature nodes; temperatures T_1 and T_2 for the two rooms, T_c for the concrete slab, T_o for the building exterior, and T_w for the supply water. The latter two temperatures are assumed to be constant. If an uncertain amount of heat $q_i + \Delta q_i$ is added to room i, the dynamics of mode 1 when the pump is turned on are written as follows:

$$\begin{aligned} C_c \dot{T}_c &= \textstyle\sum_{i=1}^{2} \frac{1}{R_{ic}} \left(T_i - T_c\right) + \frac{1}{R_{cw}} \left(T_w - T_c\right), \\ C_1 \dot{T}_1 &= \frac{1}{R_{1c}} \left(T_c - T_1\right) + \frac{1}{R_{1o}} \left(T_o - T_1\right) + \frac{1}{R_{12}} \left(T_2 - T_1\right) + q_1 + \Delta q_1, \\ C_2 \dot{T}_2 &= \frac{1}{R_{2c}} \left(T_c - T_2\right) + \frac{1}{R_{2o}} \left(T_o - T_2\right) + \frac{1}{R_{12}} \left(T_1 - T_2\right) + q_2 + \Delta q_2. \end{aligned} \tag{23}$$

The dynamics when the water pump is turned off (mode 2) are identical to those of mode 1 but with $R_{cw} = +\infty$. Thermal capacitance is easy to compute for a solid material like

Table 1 Parameter values for hydronic heating

Room 1 floor dimensions	$A_{1c} = 8\,m \times 6\,m$	$R_{1c} = \frac{r_{1c}}{A_{1c}} = 0.0026\frac{K}{W}$
Room 1 area to outside	$A_{1o} = 24\,m^2$	$R_{1o} = \frac{r_{1o}}{A_{1o}} = 0.0491\frac{K}{W}$
Room 2 floor dimensions	$A_{2c} = 6\,m \times 6\,m$	$R_{2c} = \frac{r_{2c}}{A_{2c}} = 0.0035\frac{K}{W}$
Room 2 area to outside	$A_{2o} = 18\,m^2$	$R_{2o} = \frac{r_{2o}}{A_{2o}} = 0.0654\frac{K}{W}$
Interconnecting wall area	$A_{12} = 18\,m^2$	$R_{12} = \frac{r_{12}}{A_{12}} = 0.0439\frac{K}{W}$
Total slab area	$A_c = 42\,m^2$	$R_{cw} = \frac{r_{wc}}{A_c} = 0.0012\frac{K}{W}$
Room 1 volume	$V_1 = 144\,m^3$	$C_1 = 5c_a\rho_a V_1 = 8.774 \times 10^5\frac{J}{K}$
Room 2 volume	$V_2 = 108\,m^3$	$C_2 = 5c_a\rho_a V_2 = 6.580 \times 10^5\frac{J}{K}$
Slab volume	$V_c = 10.5\,m^3$	$C_c = c_c\rho_c V_c = 3.623 \times 10^7\frac{J}{K}$
Nominal heat gains	$q_1 = \left(6\frac{W}{m^2}\right) \times A_{1c}$	$q_2 = \left(8\frac{W}{m^2}\right) \times A_{2c}$

concrete;[11] for the rooms we compute the thermal capacitance from air volume and compensate for furniture and other objects by multiplying with a factor 5 (Sourbron et al. 2013). We use formulas and standard parameters from (Gwerder et al. 2008) to compute thermal heat insulation coefficients[12] r_{ij} which give the thermal resistance as $R_{ij} = r_{ij}/A_{ij}$ for the corresponding wall or floor area. The inner wall heat insulation coefficient is taken to be $r_{12} = 0.79\,m^2K/W$ (Sourbron et al. 2013). In Table 1 the dimensions of the zones are listed along with the associated resistances and capacitances.

We assume that the outdoor temperature is $T_o = 30\,°\text{C}$, that the water temperature is $T_w = 18\,°\text{C}$, and that the uncertainty is bounded as $|\Delta q_i| \leq \left(3\frac{W}{m^2}\right) \times A_i$ for $i = 1, 2$. We set the domain to $20 \leq T_c, T_1, T_2 \leq 28$ and consider as objective to control the room temperatures to a desired range, denoted by a proposition SET ($21 \leq T_c \leq 27$ and $22 \leq T_1, T_2 \leq 25$), and guarantee invariance in this range, captured by the LTL formula $\varphi_2 := \Diamond\Box SET$. The problem is under-actuated and the equilibrium points of Eq. 23 are outside of SET, so it does not have a trivial solution.

Applying the abstraction-synthesis-refinement loop yields a winning set that covers 76% percent of the domain after 1200 iterations, which takes about 45 minutes on a modern desktop computer. After the winning set has been found we refine the part represented by $\text{Win}_{\exists,\forall}(\Box SET)$ further to eliminate Zenoness of the controller. Figure 6 shows a simulation trace of the corresponding controller under disturbance, and a 3D illustration of trajectories converging to the target set.

The example above requires using multi-mode progress groups to compute a reasonably sized winning set. Indeed, the approach in (Sun et al. 2014) fails to find a winning set for this problem for relatively large uniform abstractions. For comparison, we also solve the same problem without disturbance and with the parameter set from (Sun et al. 2014). That paper proposes efficient algorithms for synthesis; the bottle neck is creating the abstraction which for this problem takes 4.5 hours for an abstraction with 4000 states. Solving the synthesis problem on the resulting abstraction gives a winning set that covers 63% of the domain. In contrast, with our abstraction-synthesis-refinement loop we can on the same computer incrementally construct a non-uniform abstraction with a winning set that covers 64% of

[11] The formula is $c\rho V$, where c is specific heat $[J/kgK]$, ρ is density $[kg/m^3]$, and V is volume $[m^3]$.

[12] $r_{1o} = r_{2o} = 1.178\,m^2K/W$, $r_{1c} = r_{2c} = 0.125\,m^2K/W$, $r_{cw} = 0.102\,m^2K/W$.

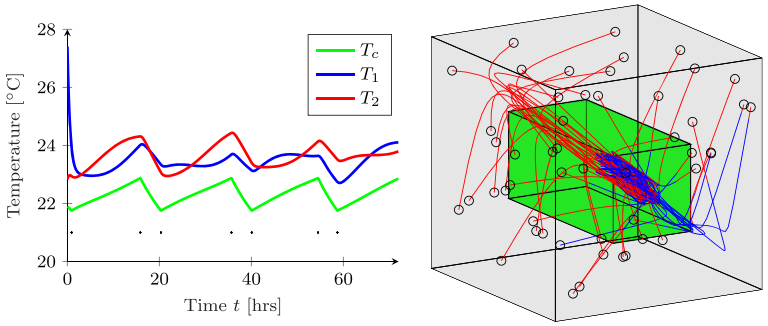

Fig. 6 *Left*: Plot of temperatures versus time when each room is subject to an independent sinusoidal disturbance that models varying utilization of the room throughout the day. As can be seen, the temperatures converge to the desired ranges and remain there. The *black dots* at the *bottom* indicate switches. *Right*: Plot showing how 50 sample trajectories converge to the target SET (*green*) starting from different parts of the winning set. Different trajectory colors correspond to different control actions

the domain in 1 minute. The abstraction has 270 states at this stage which shows the benefit of the incremental and non-uniform approach. With additional iterations it is possible to further enlarge the winning set to cover 83% of the domain.

10 Conclusions

This paper treated augmented finite transition systems (AFTS) and described how they can be used to construct abstractions of continuous-state switched systems, both in discrete and continuous time. The advantage of AFTSs over standard finite transition systems is the possibility to encode liveness properties of the continuous dynamics in progress groups, which eliminates spurious trajectories in the abstraction and thus provides a tighter over-approximation. The procedure for constructing an abstracting AFTS hinges upon the implementation of two subroutines `isBlocked` and `isTransient`, for which we gave convex optimization-based implementations in two important cases: linear and polynomial dynamics.

We described how synthesis problems for general LTL specifications can be solved on an AFTS, and proposed an abstraction-synthesis-refinement loop which refines an over-approximating AFTS until a satisfactory solution—or a counter-example—is found. We also gave efficient AFTS synthesis algorithms for an important fragment of LTL. The benefits of using these algorithms include incremental synthesis and localized partition refinement, both reducing the required computational effort compared to using general-purpose synthesis tools. We suggested methods to eliminate Zeno behavior in continuous-time systems when implementing a switching protocol obtained from an abstraction, and illustrated our approach on two examples.

Current work focuses on identification of classes of nonlinear dynamical systems for which the `isBlocked` and `isTransient` subroutines can be implemented even more efficiently compared to the positive polynomial-based approach considered in this work (Yang et al. 2016).

Acknowledgments The authors would like to thank Pavithra Prabhakar, Fei Sun, Eric M. Wolff and Richard M. Murray for useful discussions at the early stages of this work. Petter Nilsson is supported by NSF grant CNS-1239037. Necmiye Ozay is supported in part by NSF grants CNS-1446298 and ECCS-1553873, and DARPA grant N66001-14-1-4045. Jun Liu is supported in part by NSERC Canada.

Appendix A: Synthesis-related proofs

Lemma 1 *Algorithm* (12) *is sound and complete.*

Proof For soundness, we first show that a trajectory starting in $q \in X_{k+1} \setminus X_k$ can be steered to X_k in finite time while remaining in B. There are two cases. First, if $q \in B \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}(X_k) \setminus X_k$, then X_k can be reached in one time step by definition of $\mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}$. Secondly, if $q \in \mathrm{PGPre}_{\sharp,\forall}(X_k, B) \setminus X_k$ then there exists (G, U) with $G \in \mathcal{G}(U)$ such that X_k can be reached by keeping the state inside of $G \cap B$ using actions in U until a transition to X_k occurs due to the progress property (if $\sharp = \forall$, then $U = \mathcal{U}$ which enables progress for uncontrollable modes). This shows that $X_{k+1} \subset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(B \,\mathbf{U}\, X_k)$. By induction the soundness of the algorithm follows.

For completeness, we show that if a state q is not in X_∞, it is not in the $(\sharp, \forall)$-winning set of $B \,\mathbf{U}\, Z$. Assume by contradiction that q is in the $(\sharp, \forall)$-winning set of $B \,\mathbf{U}\, Z$; that is, from q there exists a control strategy such that Z is reached in finitely many steps during which B holds. This can happen in two different ways. First, there exists a bound K on the number of steps within which Z is guaranteed to be reached. In this case, q is in $\tilde{X}_K$ for $\tilde{X}_0 = Z$, $\tilde{X}_{k+1} = B \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}(\tilde{X}_k)$, which contradicts to the fact that X_∞ is a fixed point that does not contain q due to the $B \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}(X_k)$ term in Eq. 12. Secondly, Z is guaranteed to be reached from q while remaining in B but no control strategy can guarantee a bound on the number of steps. In this case, the controlled trajectories are not guaranteed to avoid a progress group. Without loss of generality we can take q to be in the last progress group G for some action set U before reaching X_∞ as otherwise the prefix part can be handled by the arguments in the first case and inductively applying the arguments for the second case. But then $q \in \mathrm{Inv}^{U,G}_{\sharp,\forall}(X_\infty, B) \subset \mathrm{PGPre}^{\mathcal{G}}_{\sharp,\forall}(X_\infty, B)$, which again contradicts to the fact that X_∞ is a fixed point. □

Lemma 2 *Algorithm* (14) *is sound and complete.*

Proof The following two LTL identities will be used below:

$$\Box\left(\psi_1 \wedge \left(\bigwedge_{i\in I} \Diamond \psi_2^i\right)\right) = \Box\psi_1 \wedge \left(\bigwedge_{i\in I} \Box\left(\mathtt{True}\ \mathbf{U}\ \psi_2^i\right)\right)$$
$$= \Box\left[\bigwedge_{i\in I}\left[\psi_1\ \mathbf{U}\ \left(\psi_1 \wedge \psi_2^i\right)\right]\right], \tag{24}$$

$$(\psi_3\ \mathbf{U}\ \psi_1) \vee (\psi_3\ \mathbf{U}\ \psi_2) = \psi_3\ \mathbf{U}\ (\psi_1 \vee \psi_2). \tag{25}$$

Denote the specification by φ_1, i.e., $\varphi_1 := (B \,\mathbf{U}\, Z) \vee \Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond C^i\right)\right)$. We call a specification ψ_1 *stronger* than ψ_2 if for any word w, $w \models \psi_1 \implies w \models \psi_2$. If ψ_1 is stronger than ψ_2, then $\mathrm{Win}(\psi_1) \subset \mathrm{Win}(\psi_2)$.

To prove soundness, we remark that any fixed point $\overline{W}$ of Eq. 14 satisfies

$$\overline{W} = \bigcap_{i\in I} \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(B \mathbf{U} \left(Z \cup \left(B \cap C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\overline{W}\right)\right)\right)\right).$$

Consider $q \in \overline{W}$, a trajectory starting in q can for each i be controlled to reach either Z or $B \cap C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\overline{W}\right)$ while remaining in B. If the former occurs, $B \mathbf{U} Z$, and thus φ_1, is evidently satisfied by the trajectory. If the latter occurs, the set $B \cap C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\overline{W}\right)$ is eventually reached. Because of the $\mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\overline{W}\right)$ term, the argument repeats which shows that either $B \mathbf{U} Z$ or $\Box\left(B \mathbf{U} \left(B \cap C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\overline{W}\right)\right)\right) = \Box\left(B \wedge \Diamond\left(C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}(\overline{W})\right)\right)$ is satisfied by the trajectory (the equality follows from Eq. 24). This holds for any i, which is a stronger condition than φ_1. Thus the algorithm is sound.

We prove completeness, which amounts to showing that $\mathrm{Win}(\varphi_1) \subset W_\infty$. Consider the specification

$$\varphi_2 := \bigwedge_{i\in I}\left[B \mathbf{U} \left(Z \cup \left(B \cap C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)\right)\right)\right)\right].$$

Since φ_1 is a liveness specification, a trajectory satisfying φ_1 must remain in $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)$. Therefore the winning set of φ_1 is equal to the winning set of the specification

$$(B \mathbf{U} Z) \vee \Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond\left(C^i \cap \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)\right)\right)\right).$$

Using Eq. 24 above, this is in turn equal to

$$(B \mathbf{U} Z) \vee \Box\left(\bigwedge_{i\in I}\left[B \mathbf{U} \left(B \cap C^i \cap \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)\right)\right]\right),$$

which is stronger than

$$\begin{aligned}(B \mathbf{U} Z) \vee \left(\textstyle\bigwedge_{i\in I}\left[B \mathbf{U} \left(B \cap C^i \cap \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)\right)\right]\right)\\ = \textstyle\bigwedge_{i\in I}\left[(B \mathbf{U} Z) \vee \left(B \mathbf{U} \left(B \cap C^i \cap \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)\right)\right)\right].\end{aligned}$$

By Eq. 25 and since $B \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)\right) = B \cap \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)$, this is stronger than φ_2. Thus $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1) \subset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_2)$.

Take any set K such that $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1) \subset K$. The specification φ_2 is stronger than the specification

$$\varphi_K := \bigwedge_{i\in I}\left[B \mathbf{U} \left(Z \cup \left(B \cap C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}(K)\right)\right)\right],$$

which implies that $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_2) \subset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_K)$. Furthermore, in general $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi_1 \wedge \psi_2) \subset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi_1) \cap \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi_2)$, which implies

$$\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_K) \subset T_K := \bigcap_{i\in I} \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(B \mathbf{U} \left(Z \cup \left(B \cap C^i \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}(K)\right)\right)\right).$$

We recognize the right-hand side from Eq. 14 and conclude that if $q \in \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1)$, then the inclusions $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1) \subset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_2) \subset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_K)$ imply that q is never excluded during the algorithm (14) since it consists of iterating the mapping $K \mapsto T_K$. Thus, $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\varphi_1) \subset W_\infty$, which proves completeness. □

Lemma 3 *Algorithm* (15) *is sound and complete.*

Proof We first prove soundness and completeness for $\Box A = \texttt{True}$. Thus let $\psi := \Diamond\Box B \wedge \left(\bigwedge_{i\in I} \Box\Diamond C^i\right)$. We first show that the algorithm is sound. The specification $\Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond C^i\right)\right)$ is stronger than ψ, therefore it follows that $V_1 \subset \text{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi)$. Consider V_k for $k > 1$. Starting anywhere in V_k, the state can either be controlled to satisfy $B \,\mathbf{U}\, \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(V_{k-1})$ or $\Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond C^i\right)\right)$. If the former happens, the state can be controlled to V_{k-1} and an induction argument over k completes the soundness proof.

For completeness, consider the set $\text{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi) \setminus V_\infty$, where V_∞ is the smallest fixed point of Eq. 15. From any $q_1 \in \text{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi) \setminus V_\infty$, the state can *not* be controlled to V_∞, otherwise q_1 would have been included in V_∞ by an argument analogous to the completeness proof of Lemma 1. By repeating the argument, we conclude that an infinite trajectory in $\text{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi) \setminus V_\infty$ can be generated that satisfies the specification ψ. By considering an appropriate suffix of such a trajectory, this implies the existence of $q_2 \in \text{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi) \setminus V_\infty$ from where the specification $\Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond C^i\right)\right)$ can be enforced. But this is a contradiction since such a q_2 is in V_1 and hence in V_∞.

We finally incorporate the $\Box A$ term and show necessity and sufficiency of the restriction to V_{inv}. As established above, the algorithm is sound and complete for ψ. All fixed points are ultimately defined in terms of the winning set of $B \,\mathbf{U}\, Z$, so amending $\Box A$ to the top level specification ψ propagates that term down to the "until" level. It is therefore necessary and sufficient to replace each computation of $\text{Win}^{\mathcal{T}}_{\sharp,\forall}(B \,\mathbf{U}\, Z)$ with $\text{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box A \wedge (B \,\mathbf{U}\, Z))$. Since $V_{inv} = \text{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box A)$, it follows that $\text{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box A \wedge (B \,\mathbf{U}\, Z)) = \text{Win}^{\mathcal{T}}_{\sharp,\forall}((B \cap V_{inv}) \,\mathbf{U}\, (Z \cap V_{inv}))$ which shows the correctness of the restriction technique. □

Lemma 4 *Algorithm* (16) *is sound and complete.*

Proof For soundness, consider a set of fixed points $\overline{X}^J$ for $J \in 2^I$ that satisfy

$$\overline{X}^J = Z \cup \left(\left(\bigcap_{i\in J} B^i\right) \cap \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}\left(\bigcup_{K\in 2^J} \overline{X}^K\right)\right).$$

Let $q \in \overline{X}^J$. If $q \in Z$ the specification is evidently satisfied. Otherwise, $q \in \left(\bigcap_{i\in J} B^i\right) \cap \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}\left(\bigcup_{K\in 2^J} \overline{X}^K\right)$ which implies that q can be controlled to $\overline{X}^{K_1}$ for some $K_1 \subset J$. If Z is not reached, an induction argument results in a chain $J \supset K_1 \subset K_2 \supset \ldots$ which necessarily converges to some non-empty subset K_∞ of J. Thus there is a strategy that eventually enforces $\Box B^i$ for some $i \in J$.

For completeness, remark that $X^{\{i\}}_1 = Z \cup B^i$. We show that if $q_0 \notin X^J_{k_0+1}$ but $q_0 \in X^J_{k_0}$, then nondeterminism can force a trajectory starting in q_0 to avoid $X^{\{i\}}_1$ for all $i \in I$ while also avoiding Z. To this end, assume that $q_0 \in X^J_{k_0} \setminus X^J_{k_0+1}$. Evidently, $q_0 \notin Z$, which implies that $q_0 \in \bigcap_{i\in J} B^i$ and $q_0 \notin \text{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}\left(\bigcup_{K\in 2^J} X^K_{k_0}\right)$. The latter means that nondeterminism can prevent a transition to $X^K_{k_0}$ for any $K \subset J$. Assume a transition to q_1 occurs and that q_1 was excluded from $\{X^K_k\}$ for $k = k_1$. Induction over time and set inclusion results in a strictly decreasing sequence $k_0 k_1, \ldots$ and a trajectory $q_0 q_1 \ldots$ where

$q_t \notin \bigcup_{K\in 2^J} X^K_{k_t}$. In particular, $q_t \notin Z$ and there exists a finite T s.t. $q_T \notin B^i$ for any $i \in J$, which shows completeness of the algorithm. □

Lemma 5 *Algorithm* (17) *is sound and complete.*

Proof For soundness, we remark that $W_1 = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\bigvee_{i\in I}\left[\left(B^i \mathbf{U}\, Z\right) \vee \Box B^i\right]\right)$ which is contained in $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Diamond Z \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)\right)$. Starting in W_k for $k > 1$, a strategy exists that either results in $\Box B^i$ being fulfilled for some i, or such that W_k can be reached. An induction argument completes the soundness proof.

For completeness, assume that $q \in \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Diamond Z \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)\right) \setminus W_\infty$ where W_∞ is the smallest fixed point of Eq. 17. Since $\mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}(W_\infty) \cup \mathrm{PGPre}^{\mathcal{T}}_{\sharp,\forall}(W_\infty, Q) \subset W_\infty$, it follows by the completeness argument in the proof of Lemma 1 that a transition to W_∞ can never be enforced for a trajectory starting in q. Since $Z \subset W_\infty$, it follows that nondeterminism can generate a trajectory that never enters W_∞ and which satisfies $\Diamond\Box B^i$ for some i. Taking a suffix of such a trajectory, it follows that there exists $\tilde{q} \in \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Diamond Z \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)\right) \setminus W_\infty$ from where $\bigvee_{i\in I} \Box B^i$ can be enforced. But then $\tilde{q} \in W_1 \subset W_\infty$ which is a contradiction. Thus (17) is complete. □

Lemma 6 *Algorithm* (18) *is sound and complete.*

Proof Let $\psi := \Diamond A \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right) \vee \Box\Diamond C$. For soundness, consider a fixed point $\overline{V}$ of (18). It has the property

$$\overline{V} = \mathrm{Win}_{\sharp,\forall}\left(\Diamond\left(A \cup \left(C \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\overline{V}\right)\right)\right) \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)\right).$$

Starting in $\overline{V}$, there is a strategy to ensure one of $\psi_1 := \Diamond A$, $\psi_2 := \bigvee_{i\in I} \Diamond\Box B^i$, or $\psi_3 := \Diamond\left(C \cap \mathrm{Pre}^{\mathcal{T}}_{\sharp,\forall}\left(\overline{V}\right)\right)$. If ψ_1 or ψ_2 occur, ψ is evidently satisfied. If ψ_3 occurs, an induction argument shows that $\psi_1 \vee \psi_2 \vee \Box\psi_3 = \psi$ holds.

For completeness, remark that any trajectory that enforces ψ must remain in $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi)$ due to ψ being a liveness property. Furthermore, $\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}\left(\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi)\right) = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi)$ since ψ can be enforced from its pre-image. Therefore,

$$\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi) = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\tilde{\psi}\right), \quad \text{for} \quad \tilde{\psi} := \Diamond A \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right) \vee \Box\Diamond \times \left(C \cap \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}\left(\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi)\right)\right).$$

Take any $K \supset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi)$ and let $\psi_K := \Diamond A \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right) \vee \Diamond\left(C \cap \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\forall}(K)\right)$. It can be seen that $\tilde{\psi}$ is stronger than ψ_K, which implies the inclusion $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi) \subset \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi_K)$. This shows that elements in $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi)$ are never excluded in Eq. 18, which consist of iterations of the mapping $K \mapsto \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\psi_K)$. □

Appendix B: Candidate set derivation

Let us call an algorithm expanding or contracting if the sequence $\{V_k\}_{k\geq 1}$ of intermediate sets it computes is enlarging or shrinking, respectively. All fixed point algorithms considered in this paper are either contracting or expanding. Moreover, their output is monotone with respect to initial conditions (i.e., a larger initial set gives a larger fixed point w.r.t. set inclusion). We consider sets of type $\mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\cdot)$ computed as the fixed point V_∞ of a sequence $\{V_k\}_{k\geq 1}$ and determine a candidate set C recursively according to the following rules:

– For an expanding algorithm with input V_1 and fixed point V_∞:

  Addition: add $\mathrm{Pre}_{\sharp,\exists}(V_\infty) \setminus V_\infty$ to the candidate set C.
  Recursion: if V_1 is an output of another fixed point algorithm, add its candidate set to C.

– For a contracting algorithm with input V_1 and fixed point V_∞:

  Addition: add $V_1 \setminus V_\infty$ to the candidate set C.
  Recursion: if V_1 is an output of another fixed point algorithm, add its candidate set to C.

The rationales behind these rules are as follows. Firstly, the fixed point may always be enlarged by starting with a larger initial set, thus we pursue this objective in both cases. For an expanding algorithm, we also add states adjacent to its fixed point V_∞ in the hope that a refinement may reveal states that can enlarge it further. There is no need to consider smaller sets V_i since these are all contained in V_∞. For a contracting algorithm we add $V_1 \setminus V_\infty$ in the hope that refinement may reveal control options that allow the fixed point V_∞ to be enlarged.

There are ways to further tune the candidate sets that may be suitable for certain problems. Firstly, the progress group reachability operator PGPre is computed with a contracting algorithm whose candidate set could be added to the overall candidate set. Below we disregard this potential addition in the interest of keeping the notation relatively simple; the best way to implement candidate set computation algorithmically is to follow the recursive rules above. Secondly, it may be possible to exclude parts of the candidate set of a contracting algorithm in case there are states that will for sure be excluded by the algorithm even after refinement.

We now apply the rules above to the computation of the winning set (15) to obtain a candidate set of a specification of type $\Box A \wedge \Diamond\Box B \wedge \left(\bigwedge_{i\in I} \Box\Diamond C^i\right)$. The algorithm is expanding and produces an increasing set sequence $\{V_k\}_{k\geq 1}$. We therefore do the following:

1A. Addition: add $\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}(V_\infty) \setminus V_\infty$ to the candidate set,
1R. Recursion: add candidate set of $V_1 = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Box\left(B \wedge \left(\bigwedge_{i\in I} \Diamond C^i\right)\right)\right)$ to the candidate set.

We recursively consider V_1 which is computed from algorithm (14) as the stable point W_∞ of a contracting set sequence $\{W_k\}_{k\geq 1}$. Therefore we do the following

2A. Addition: add $W_1 \setminus W_\infty = W_1 \setminus V_1$ to the candidate set,

2R. Recursion: add candidate set of $W_1 = \bigcap_{i\in I} W_{1,i}$ for $W_{1,i} := \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(B \mathbf{U} \left(B \cap C^i\right)\right)$ to the candidate set.

Next we turn to the different $W_{1,i}$'s which are each computed from Eq. 12 through an expanding set sequence $\{X_{k,i}\}_{k\geq 1}$. Since $X_{1,i} = B \cap C^i$ is not itself a fixed point, the recursion stops here and we arrive at:

3A. Addition: add $B \cap \left(\bigcup_{i\in I} \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(W_{1,i}\right) \setminus W_{1,i}\right)$ to the candidate set. Here we have intersected with B since states in B^C can not be candidates to $B \mathbf{U} (B \cap C^i)$.

However, there is one final part of the overall candidate set, since the winning set computation is restricted to $V_{inv} = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box A)$. This set is computed using Eq. 14 as the convergence value of a contracting set sequence $\{\tilde{W}_k\}_{k\geq 1}$ with $\tilde{W}_1 = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(A \mathbf{U} A) = A$. Therefore we make a final addition to the candidate set:

4A. Addition: add $A \setminus V_{inv}$ to the candidate set.

Collecting the different pieces from above, we arrive at the following candidate set:

$$\begin{aligned} C_{\sharp,\forall}\left(\Box A \wedge \Diamond\Box B \wedge \left(\textstyle\bigwedge_{i\in I} \Box\Diamond C^i\right)\right) = &\left(\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(V_\infty\right) \setminus V_\infty\right) \cup \left(W_1 \setminus V_1\right) \\ &\cup \left(B \cap \left(\textstyle\bigcup_{i\in I} \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(W_{1,i}\right) \setminus W_{1,i}\right)\right) \cup \left(A \setminus V_{inv}\right). \end{aligned} \tag{26}$$

A candidate set for the dual algorithm (18) can be derived in a similar way. Let $\{V_k\}_{k\geq 1}$ be the contracting set sequence generated by Eq. 18.

1A. Addition: add $V_1 \setminus V_\infty$ to the candidate set,
1R. Recursion: add candidate set of $V_1 = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Diamond\left(A \cup C\right) \vee \left(\bigvee_{i\in I} \Diamond\Box B^i\right)\right)$ to the candidate set.

Proceeding with V_1, it is equal to the fixed point value W_∞ of an expanding sequence $\{W_k\}_{k\geq 1}$ generated by Eq. 17.

2A. Addition: add $\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(W_\infty\right) \setminus W_\infty = \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(V_1\right) \setminus V_1$ to the candidate set,
2R. Recursion: add candidate set of $W_1 = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\bigvee_{i\in I} [\Box B_i \vee (B_i \mathbf{U} (A \cup C))]\right)$ to the candidate set.

Finally, W_1 is equal to the union over the set of fixed points X^J_∞ of the contracting algorithm (16). At the first iteration, $X^J_1 = A \cup C \cup \left(\bigcap_{i\in J} B^j\right)$. We therefore get one final addition to the candidate set as:

2A. Addition: add $\bigcup_{J\in 2^I} A \cup C \cup \left(\bigcap_{i\in J} B^i\right) \setminus W_1$ to the candidate set. However, it holds that $\bigcup_{J\in 2^I} \bigcap_{i\in J} B^i = \bigcup_{i\in I} B^i$ and furthermore $A \cup C \in W_1$. Therefore the additional set simplifies to $\bigcup_{i\in I} B^i \setminus W_1$.

Combined, we arrive at the following candidate set:

$$C_{\sharp,\forall}\left(\Diamond A \bigvee_{i\in I} \Diamond\Box B^i \vee \Box\Diamond C\right) = \left(V_1 \setminus V_\infty\right) \cup \left(\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(V_1\right) \setminus V_1\right) \cup \left(\textstyle\bigcup_i B^i \setminus W_1\right). \tag{27}$$

For illustration purposes, we consider two notable special cases of Eq. 26. First, for a specification of the form $\Diamond\Box B$, the expression simplifies to

$$C_{\sharp,\forall}\left(\Diamond\Box B\right) = \left(\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(V_\infty\right) \setminus V_\infty\right) \cup \left(B \setminus V_1\right), \tag{28}$$

for $V_\infty = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\Diamond\Box B)$ and $V_1 = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}(\Box B)$. The same expression can also be obtained from Eq. 27.

Secondly, for a specification of the form $\bigwedge_{i\in I}\Box\Diamond C^i$, we obtain from Eq. 26 that

$$C_{\sharp,\forall}\left(\bigwedge_{i\in I}\Box\Diamond C^i\right) = \left(\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}(V_\infty)\setminus V_\infty\right)\cup\left(\bigcap_{i\in I}W_{1,i}\setminus V_\infty\right)\cup \times\left(\bigcup_{i\in I}\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(W_{1,i}\right)\setminus W_{1,i}\right),$$

where $V_\infty = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\bigwedge_{i\in I}\Box\Diamond C^i\right)$ and $W_{1,i} = \mathrm{Win}^{\mathcal{T}}_{\sharp,\forall}\left(\Diamond C^i\right)$. However, the expression can be simplified further since the first term is contained in the union of the last two:[13]

$$C_{\sharp,\forall}\left(\bigwedge_{i\in I}\Box\Diamond C^i\right) = \left(\bigcap_{i\in I}W_{1,i}\setminus V_\infty\right)\cup\left(\bigcup_{i\in I}\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(W_{1,i}\right)\setminus W_{1,i}\right). \tag{29}$$

Again, the same expression can be obtained from Eq. 27 for the special case $I = \{1\}$.

References

Ahmadi AA, Majumdar A (2014) DSOS and SDSOS optimization: LP and SOCP-based alternatives to sum of squares optimization. In: Proceedings of the IEEE conference on decision and control, pp 394–401

Ahmadi AA, Parrilo PA (2014) Towards scalable algorithms with formal guarantees for Lyapunov analysis of control systems via algebraic optimization

Baier C., Katoen J. (2008) Principles of model checking, MIT press

Batt G., Belta C., Weiss R. (2008) Temporal logic analysis of gene networks under parameter uncertainty. IEEE Trans Automatic Control 53(Special Issue):215–229

Belta C., Habets L. (2006) Controlling a class of nonlinear systems on rectangles. IEEE Trans Automatic Control 51(11):1749–1759

Bloem R, Jobstmann B, Piterman N, Pnueli A, Saar Y (2012) Synthesis of reactive (1) designs. J Comput System Sci 78:911–938

Cámara J, Girard A, Gössler G (2011) Synthesis of switching controllers using approximately bisimilar multiscale abstractions. In: Proceeding of HSCC, pp 191–200

Church A (1962) Logic, arithmetic and automata. In: Proceedings of the international congress of mathematicians, pp 23–35

Clarke FH, Ledyaev YS, Stern RJ, Wolenski PR (1998) Nonsmooth analysis and control theory. Springer

Coogan S, Arcak M (2015) Efficient finite abstraction of mixed monotone systems. In: Proceedings of HSCC, pp 58–67

Feuer A, Heymann M (1976) Ω-invariance in control systems with bounded controls. J Math Anal Appl 53(2):266–276

Filippidis I, Dathathri S, Livingston SC, Ozay N, Murray RM (2016) Control design for hybrid systems with TuLiP: The temporal logic planning toolbox. In: Proceedings of MSC

Girard A, Martin S (2012) Control synthesis for constrained nonlinear systems using hybridization and robust controllers on simplices. IEEE Trans Automatic Control 57:1046–1051

Girard A, Pola G, Tabuada P (2010) Approximately bisimilar symbolic models for incrementally stable switched systems. IEEE Trans Automatic Control 55(1):116–126

Gol E, Ding X, Lazar M, Belta C (2012) Finite bisimulations for switched linear systems. In: Proceedings of IEEE CDC, pp 7632–7637

[13] Assume that $q \in \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}(V_\infty)\setminus V_\infty$. Since $V_\infty \subset W_{1,i}$ for all i, it follows that $q \in \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}(V_\infty) \subset \bigcap_{i\in I}\mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(W_{1,i}\right)$. Assume $q \notin \mathrm{Pre}^{\mathcal{T},\mathcal{U}}_{\sharp,\exists}\left(W_{1,i}\right)\setminus W_{1,i}$ for any $i \in I$, then evidently $q \in W_{1,i}$.

Grädel E, Thomas W, Wilke T (eds.) (2002) Automata, Logics, and Infinite Games: A Guide to Current Research, Lecture Notes in Computer Science, vol. 2500. Springer
Gwerder M, Lehmann B, Tödtli J, Dorer V, Renggli F (2008) Control of thermally-activated building systems (tabs). Appl Energy 85(7):565–581. doi:http://dx.doi.org/10.1016/j.apenergy.2007.08.001
Habets L., Collins P., van Schuppen J. (2006) Reachability and control synthesis for piecewise-affine hybrid systems on simplices. IEEE Trans Automatic Control 51(6):938–948
Jiang ZP, Wang Y (2002) A converse Lyapunov theorem for discrete-time systems with disturbances. Syst Control Lett 45(1):49–58
Kesten Y, Pnueli A (2000) Verification by augmented finitary abstraction. Inf Comput 163(1):203–243
Lin Y, Sontag ED, Wang Y (1996) A smooth converse Lyapunov theorem for robust stability. SIAM J Control Optimization 34(1):124–160
Liu J, Ozay N, Topcu U, Murray R (2013) Synthesis of reactive switching protocols from temporal logic specifications. IEEE Trans Automatic Control
Löfberg J (2004) Yalmip: a toolbox for modeling and optimization in matlab. In: Proceeding of IEEE CACSD, pp 284–289
Mattila R, Mo Y, Murray RM (2015) An iterative abstraction algorithm for reactive correct-by-construction controller synthesis. In: Proceedings of IEEE CDC, pp 6147–+6152
Nilsson P, Ozay N (2014) Incremental synthesis of switching protocols via abstraction refinement. In: Proceedings of CDC, pp 6246–6253
Ozay N, Liu J, Prabhakar P, Murray R (2013) Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems. American Control Conference
Parrilo P (2003) Semidefinite programming relaxations for semialgebraic problems. Math Program 96(2):293–320
Piterman N, Pnueli A (2006) Faster solutions of rabin and streett games. In: Proceedings of IEEE LICS, pp 275–284
Piterman N, Pnueli A, Sa'ar Y (2006) Synthesis of reactive (1) designs. In: Proceedings of VMCAI, pp 364–380
Pnueli A, Rosner R (1989) On the synthesis of an asynchronous reactive module. In: Proceedings of ICALP, pp 652–671
Romaní J, de Gracia A, Cabeza LF (2016) Simulation and control of thermally activated building systems (TABS). Energy & Buildings 127:22–42
Sourbron M, Verhelst C, Helsen L (2013) Building models for model predictive control of office buildings with concrete core activation. J Build Perform Simul 6(3):175–198
Sun F, Ozay N, Wolff EM, Liu J, Murray RM (2014) Efficient control synthesis for augmented finite transition systems with an application to switching protocols. In: Proceedings of ACC
Svorenova M, Kretinsky J, Chmelik M, Chatterjee K, Cerna I, Belta C (2015) Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. In: Proceedings of HSCC, pp 259–268
Tabuada P (2009) Verification and control of hybrid systems: a symbolic approach. Springer
Walter W, Thompson R (1998) Ordinary differential equations, 1 edn. Springer
Wolff E, Topcu U, Murray R (2013) Efficient reactive controller synthesis for a fragment of linear temporal logic. In: Proceedings of IEEE ICRA, pp. 5033–5040
Yang L, Ozay N, Karnik A (2016) Synthesis of fault tolerant switching protocols for vehicle engine thermal management. In: Proceedings of ACC, pp 4213–4220
Yordanov B, Tumova J, Cerna I, Barnat J, Belta C (2012) Temporal logic control of discrete-time piecewise affine systems. IEEE Trans Automatic Control 57(6):1491–1504

Petter Nilsson received his B.S. in Engineering Physics in 2011, and his M.S. in Optimization and Systems Theory in 2013, both from KTH Royal Institute of Technology in Stockholm, Sweden. While obtaining these degrees, he spent one year at École Polytechnique in Paris, France, and was also a visiting researcher at California Institute of Technology in Pasadena, CA. In addition to his technical degrees, he holds a B.S. in Business and Economics from Stockholm School of Economics in Stockholm, Sweden.

He is currently a Ph.D. candidate in Electrical Engineering: Systems at the University of Michigan, Ann Arbor, supervised by Jessy W. Grizzle and Necmiye Ozay. His research is focused on developing formal control algorithms for safety-critical cyber-physical systems.

Necmiye Ozay received the B.S. degree from Bogazici University, Istanbul in 2004, the M.S. degree from the Pennsylvania State University, University Park in 2006 and the Ph.D. degree from Northeastern University, Boston in 2010, all in electrical engineering. She was a postdoctoral scholar at California Institute of Technology, Pasadena between 2010 and 2013. She is currently an assistant professor of Electrical Engineering and Computer Science, at the University of Michigan, Ann Arbor.

Dr. Ozay's research interests include dynamical systems, control, optimization and formal methods with applications in cyber-physical systems, system identification, verification and validation, and autonomy and vision. Her papers received several awards including an IEEE Control Systems Society Conference on Decision and Control Best Student Paper Award in 2008. She received a DARPA Young Faculty Award in 2014 and an NSF CAREER Award in 2016 and was selected as an Outstanding Reviewer of the IEEE Transactions on Automatic Control in 2011. She is member of the IEEE and of the IEEE Control Systems Society Technical Committee on Computational Aspects of Control System Design.

Jun Liu received the B.S. degree in Applied Mathematics from Shanghai Jiao-Tong University, Shanghai, China, in 2002, the M.S. degree in Mathematics from Peking University, Beijing, China, in 2005, and the Ph.D. degree in Applied Mathematics from the University of Waterloo, Waterloo, Canada, in 2010. He was a Lecturer in Control and Systems Engineering at the University of Sheffield from 2012 to 2015 and a Postdoctoral Scholar in Control and Dynamical Systems at the California Institute of Technology from 2011 to 2012. He is currently an Assistant Professor in Applied Mathematics at the University of Waterloo. His interests are in the theory and applications of hybrid systems and control, including formal, computational methods for control design and applications in cyber-physical systems.