A Simplex Architecture for Hybrid Systems using Barrier Certificates*

Junxing Yang[1], Md. Ariful Islam[2], Abhishek Murthy**[3], Scott A. Smolka[1], and Scott D. Stoller[1]

[1] Department of Computer Science, Stony Brook University, USA
[2] Department of Computer Science, Carnegie Mellon University, USA
[3] Philips Lighting Research North America, USA

Abstract. This paper shows how to use Barrier Certificates (BaCs) to design Simplex Architectures for hybrid systems. The Simplex architecture entails switching control of a plant over to a provably safe Baseline Controller when a safety violation is imminent under the control of an unverified Advanced Controller. A key step of determining the switching condition is identifying a *recoverable region*, where the Baseline Controller guarantees recovery and keeps the plant invariably safe. BaCs, which are Lyapunov-like proofs of safety, are used to identify a recoverable region. At each time step, the switching logic samples the state of the plant and uses bounded-time reachability analysis to conservatively check whether any states outside the zero-level set of the BaCs, which therefore might be non-recoverable, are reachable in one decision period under control of the Advanced Controller. If so, failover is initiated.
Our approach of using BaCs to identify recoverable states is computationally cheaper and potentially more accurate (less conservative) than existing approaches based on state-space exploration. We apply our technique to two hybrid systems: a water tank pump and a stop-sign-obeying controller for a car.

Keywords: Simplex architecture; Hybrid systems; Barrier certificates; Reachability; Switching logic

1 Introduction

The Simplex Architecture [20], illustrated in Fig. 1, traditionally consists of two versions of a controller, called the *advanced controller* (AC) and *baseline controller* (BC), and a physical *plant* (P). The advanced controller is designed

* This work is supported in part by AFOSR Grant FA9550-14-1-0261, NSF Grants IIS-1447549, CNS-1421893, CNS-1446832, CNS-1445770, CNS-1445770, and CCF-1414078, and ONR Grant N00014-15-1-2208. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these organizations.

** The author contributed to this research as part of his doctoral study at Stony Book University.

for maximum performance and is in control of the plant under nominal operating conditions. However, certification that the advanced controller keeps the plant state within a prescribed safety region (i.e., region of safe operation) may be infeasible, due to its complexity or adaptiveness, or because an accurate model of it is unavailable for analysis. In contrast, the baseline controller is certified to maintain safety of the plant. When the plant is under control of AC, a *decision module* (DM) periodically, with decision period Δt, monitors the state of the plant and switches the control of the plant to the baseline controller if the plant is in imminent danger (i.e., within the next decision period) of entering a state that might lead to a safety violation.

The switching condition used in the decision module is determined as follows. A state of the plant is *recoverable* if BC can take over from that state (due to a switch) and keep the plant invariably safe; in other words, the composition of P and BC, denoted $P \times BC$, when started from a recoverable state, will always remain within the safety region. An unbounded time horizon is used in the definition of recoverable states because, in general, we have no bound on how long BC needs to take corrective actions and overcome the plant's momentum (in a general sense, not limited to physical motion) toward unsafe states.

A state is *switching* if the plant, under control of AC, may enter an unrecoverable state during the next decision period, i.e., within time Δt. This definition reflects the discrete-time nature of DM. The switching condition simply checks whether the current state is switching.

Note that switching states are a subset of recoverable states which are a subset of safe states.

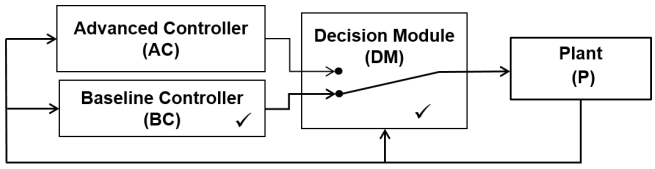


Fig. 1. The two-controller Simplex Architecture.

The earliest methodology for computing switching conditions is based on Lyapunov stability theory and reduces the problem to solving linear matrix inequalities (LMIs) [4]. The method applies to plants with linear time-invariant dynamics and a linear baseline controller [19]. This approach is computationally efficient but limited in applicability. More general approaches were later developed [3,2], based on state-space exploration, also called state-space *reachability*. Several reachability algorithms for hybrid systems have been developed, e.g., [7,21,6,1].

There are many reachability algorithms, varying in the shape of the regions (e.g., boxes or regions bounded by polynomials), whether the partitioning of the state space into regions is pre-determined or adaptive, etc.

Methods based on reachability are attractive for their broad applicability: they can handle nonlinear hybrid systems. A *hybrid system* is a system with both discrete and continuous variables and with multiple modes, each with a different dynamics (for the continuous variables). However, they face several issues. The main issue is the high computational cost associated with their reachability analysis, especially for high-dimensional systems, since the number of explored regions tends to grow exponentially with the dimensionality. While reachability algorithms are improving, the computational cost, in time or memory, remains prohibitive in many cases.

Accuracy is also an issue. Reachability algorithms generally compute an over-approximation of the reachable states. The amount of over-approximation is generally larger for non-linear systems, and it tends to increase over time (i.e., proportional to the time horizon of the reachability calculation). Reachability computations with an unbounded time horizon (as when identifying recoverable states) can in theory converge (i.e., reach a fixed-point) when the set of reachable states lies within a bounded region of the state space. In practice, however, the ever-increasing "looseness" of the over-approximation often causes the reachability computation to diverge even in those cases. Even if it converges, a loose over-approximation of reachability makes the computation of recoverable states conservative, i.e., many recoverable states will not be recognized as such, causing unnecessary switches to BC.

A third issue is the required expertise and manual effort. Reachability algorithms typically have several numerical parameters that indirectly control the cost and accuracy of the computation. While there are general guidelines and heuristics for choosing initial parameter values, detailed understanding of the reachability algorithm and the hybrid system, and considerable experimentation, are often needed to tune the parameters in order to obtain acceptably accurate results in reasonable time, when this is possible.

This paper presents an alternative approach to computing recoverable states, based on *barrier certificates* [16,17], a methodology developed for safety verification of hybrid systems. Specifically, we observe that the 0-level set of a barrier certificate for $P \times BC$ separates recoverable and unrecoverable states. We still use reachability to compute switching states. This combination of techniques—namely, using barrier certificates to compute recoverable states, and reachability to compute switching states, instead of using reachability for both—is advantageous, because the issues with reachability algorithms are much more severe when computing recoverable states, due to the unbounded time horizon, than when computing switching states, which involves a short time horizon.

Our approach is mostly automatic for a class of systems that includes some nonlinear hybrid systems. For these systems, the problem of computing a barrier certificate can be reduced to solving a sum-of-squares (SOS) optimization problem [16,17], which can be solved by semidefinite programming solvers. We use SOSTOOLS [14], a MATLAB toolbox for solving SOS optimization problems; we also experimented with Spotless (`https://github.com/spot-toolbox`). SOSTOOLS can handle hybrid systems in which the differential equations, guards,

and invariants are defined by polynomial expressions such that the minimum and maximum degrees of each polynomial are even. SOSTOOLS does require some expertise and manual effort to choose suitable parameter values.

To increase assurance in the correctness of barrier certificates computed by SOSTOOLS, we use Z3 (`https://github.com/Z3Prover/z3/wiki`), a state-of-the-art satisfiability modulo theories (SMT) solver, to verify their correctness. This ensures that bugs or limitations of numerical accuracy in SOSTOOLS cannot compromise the soundness of our results.

We apply our approach to two hybrid systems as case studies. A water tank system that controls the flow of water via a valve into a tank is modeled using a three-mode hybrid automaton with one continuous variable. The next case study describes a stop-sign-obeying controller for a car as a hybrid automaton with three-modes and two continuous variables. The implementation aspects of computing the barrier certificates are discussed and the switching logics are illustrated for the two case studies.

The rest of the paper is organized as follows. Section 2 provides background knowledge on Hybrid Systems and Barrier Certificates. Section 3 presents our approach to computing the switching logic using BaCs and bounded-time reachability. Section 4 considers our SOS characterization of BaCs. Sections 5 describes our two case studies. Section 6 discusses related work. Section 7 offers our concluding remarks and directions for future work.

2 Background

From the discussion above, we can see that the two central issues for designing the Simplex Architecture are: (1) **Identifying the Safety Region**, which results in a proof of safety of $P \times BC$, and (2) **Deriving the recoverable region and the switching boundaries**, which defines the switching logic implemented by the DM.

In this paper, we model $P \times BC$ as a hybrid system, denoted by H_B. This formalism allows us to model both the continuous-time evolution and the discrete-time instantaneous changes in the behavior of the plant under the BC's control. We formally define a hybrid system as follows.

Definition 1. *A Hybrid System $H = (\mathcal{X}, L, X_0, I, F, T)$ is a six-tuple [16]:*

- *$\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state space.*
- *L is a finite set of modes, also known as locations. The overall state space of the system is $X = L \times \mathcal{X}$ and a state of the system is denoted by $(l, \mathbf{x}) \in L \times \mathcal{X}$.*
- *$X_0 \subseteq X$ is a set of initial states.*
- *$I : L \to 2^{\mathcal{X}}$ is the invariant, which assigns to each location l an invariant set $I(l) \subseteq \mathcal{X}$ that contains all possible continuous states while in mode l.*
- *$F : X \to 2^{\mathbb{R}^n}$ is a set of vector fields. F assigns to each $(l, \mathbf{x})$ a set $F(l, \mathbf{x}) \subseteq \mathbb{R}^n$ that constrains the evolution of the continuous state as $\dot{\mathbf{x}} \in F(l, \mathbf{x})$.*
- *$T \subseteq X \times X$ is a relation that captures the discrete transitions between two modes. A transition $((l', \mathbf{x}'), (l, \mathbf{x}))$ indicates that the system can undergo a discrete (instantaneous) transition from the state $(l', \mathbf{x}')$ to the state $(l, \mathbf{x})$.*

Discrete mode-transitions occur instantaneously in time. We define *Guards* and *Reset maps* for mode-transitions as follows. $\text{Guard}(l', l) = \{\mathbf{x}' \in \mathcal{X} : ((l', \mathbf{x}'), (l, \mathbf{x})) \in T \text{ for some } \mathbf{x} \in \mathcal{X}\}$ and $\text{Reset}(l', l) : \mathbf{x}' \mapsto \{\mathbf{x} \in \mathcal{X} : ((l', \mathbf{x}'), (l, \mathbf{x})) \in T\}$, whose domain is $\text{Guard}(l', l)$.

As per [16], for computational purposes, the uncertainty in the continuous flows, defined by F, is the result of exogenous disturbance inputs such that:

$$F(l, \mathbf{x}) = \{\dot{\mathbf{x}} \in \mathbb{R}^n : \dot{\mathbf{x}} = f_l(\mathbf{x}, d) \text{ for some } d \in D(l)\}$$

where f_l is a vector field that governs the flow of the system in location l and d is a vector of disturbance inputs that take the value in the set $D(l) \subset \mathbb{R}^m$.

Trajectories or *behaviors* of H start from some initial state $(l_0, \mathbf{x}_0) \in X_0$ and evolve in continuous time as per the dynamics defined by F until the invariant, defined by I, is violated and/or a guard is enabled resulting in an instantaneous mode switch. Trajectories are obtained by concatenating the continuous evolutions and the instantaneous discrete-time jumps between the modes.

Given a set of unsafe states $X_u \subseteq X$, H is said to be *safe* if all its trajectories avoid entering X_u. We define a mapping for mode-specific unsafe states as $\text{Unsafe}(l) = \{\mathbf{x} \in \mathcal{X} : (l, \mathbf{x}) \in X_u\}$. We also define model-specific initial states as $\text{Init}(l) = \{\mathbf{x} \in \mathcal{X} : (l, \mathbf{x}) \in X_0\}$.

It is assumed that the description of the hybrid system given above is well-posed. For example, $(l, \mathbf{x}) \in X_u$ and $(l, \mathbf{x}) \in X_0$ automatically implies that $\mathbf{x} \in I(l)$, and $((l', \mathbf{x}'), (l, \mathbf{x})) \in T$ implies that $\mathbf{x}' \in I(l')$ and $\mathbf{x} \in I(l)$.

Given a set of unsafe states X_u, the safety of a hybrid system H can be proved by computing *Barrier Certificates* (BaCs) [16]. BaCs are functions that capture the following safety requirements of a hybrid system: (1) the continuous-time evolutions within the modes must ensure that the states remain safe and (2) a mode-transition $((l', \mathbf{x}'), (l, \mathbf{x}))$ from the mode l' to l must reset a safe state $(l', \mathbf{x}') \notin \text{Unsafe}(l')$ to a safe state $(l, \mathbf{x}) \notin \text{Unsafe}(l)$. Next, we introduce the formal definition of BaCs from [16].

Definition 2. *Let the hybrid system $H = (\mathcal{X}, L, X_0, I, F, T)$, the unsafe set X_u and some fixed non-negative constants $\sigma_{(l,l')}$, for all $(l, l') \in L \times L$, be given. A BaC is a collection of functions $B_l(\mathbf{x})$, for all $l \in L$, that are differentiable with respect to its argument and satisfy:*

$$B_l(\mathbf{x}) > 0 \quad \forall \mathbf{x} \in \textit{Unsafe}(l) \tag{1}$$

$$B_l(\mathbf{x}) \leq 0 \quad \forall \mathbf{x} \in \textit{Init}(l) \tag{2}$$

$$\frac{\partial B_l}{\partial \mathbf{x}}(\mathbf{x}).f_l(\mathbf{x}, d) \leq 0 \quad \forall (\mathbf{x}, d) \in I(l) \times D(l) \tag{3}$$

$$B_l(\mathbf{x}) - \sigma_{(l',l)} B_{l'}(\mathbf{x}') \leq 0 \quad \forall (\mathbf{x}, \mathbf{x}') \in \mathcal{X}^2 \textit{ such that } \mathbf{x}' \in \textit{Guard}(l', l) \textit{ and } \mathbf{x} \in \textit{Reset}(l', l)(\mathbf{x}') \tag{4}$$

Theorem 3 and Proposition 2 of [16] ensure that the existence of BaC, as defined above, proves the safety of H. Initial states are assumed to be safe (Eqs. 1 and 2). Eq. 3 dictates that the value of the BaC cannot increase along

the continuous evolution of any trajectory within a mode. Finally Eq. 4 ensures that the discrete mode transitions reset safe states to safe states. Eqs. 1 - 4 ensure that a trajectory that starts out in an initial state, and thus with a BaC value ≤ 0, can never obtain a BaC-value of > 0. Thus the zero level sets of the functions, $B_l(\mathbf{x}) = 0$, create a "barrier" between Unsafe(l) and the safe states of the mode.

3 Switching Logic

Let P be the physical plant, AC the advanced controller, BC the baseline controller, and H_B the hybrid system modeling the composition of P and BC. We use the following notations: $\mathbf{x}_{AC}(T)$ denotes the the state of the plant under control of the AC, $\mathbf{u}_{AC}(T)$ is the control input provided by the AC, and T is the discrete time. Let M be the number of modes in H_B.

Safety of P under the BC can be established by computing a BaC $\{B_1(\mathbf{x}), B_2(\mathbf{x}), ..., B_M(\mathbf{x})\}$ for H_B. The implementation aspects of computing the BaC are deferred to Sec. 4. Given a mode l of H_B, Recov(l) denotes the intersection of the interior of the zero-level set of $B_l(x)$ and $I(l)$. Note that the sets of states Recov(l) contain the initial states and are recoverable under AC.

We make the following assumptions.

1. DM samples $\mathbf{x}_{AC}(T)$ and $\mathbf{u}_{AC}(T)$ every Δt units of time.
2. The AC also works in discrete time: $\mathbf{u}_{AC}(T)$ is updated at time $(T + \Delta t)$.

The assumptions made by the switching logic are not restrictive. Knowledge of the control input allows the switching logic to become less conservative, as discussed later in the section. The assumptions can also be relaxed by assuming conservative bounds on the plant dynamics under the AC's control. The system models also assume reliable hardware, since Simplex is not intended to tolerate hardware failures.

Reachability computation is a key element of the switching logic. $Reach_{\leq \Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T))$ denotes the set of plant states reachable under the control input $\mathbf{u}_{AC}(T)$ in the time interval $[T, T + \Delta t]$. $Reach_{=\Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T))$ is the set of plant states that are reachable under the control input $\mathbf{u}_{AC}(T)$ at time $t = T + \Delta t$.

Algorithm 1 outlines the switching logic. Step 2 involves two on-the-fly reachability computations. For scalability, the reach-set computation time must be less than or equal to Δt. The online reach-set computation algorithm of [5] can handle fairly large hybrid systems. The dimension of the largest system considered for online reachability in [5] is 30. Alternatively, we can use the real-time reachability algorithm from [9]. When online reachability computation is not scalable, we can employ a combination of offline and online strategies. In the offline step, the state and input spaces are partitioned into finite regions and reach-sets for the partitions, computed apriori, are stored in a table. At run-time, given the state and the control input, the reach set of the corresponding partition is applied.

To compute the set intersections in steps 3 and 4, we can employ standard polyhedral libraries, like *PolyLib* [10]. Non-convex zero-level sets of BaCs may

Algorithm 1: DM's Switching Logic

1 Obtain the sample $(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T))$;
2 Compute $Reach_{\leq \Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T))$ and $Reach_{=\Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T)$;
3 $safety = (Reach_{\leq \Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T)) \cap X_u == \emptyset)$;
4 $recoverability = Reach_{=\Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T)) \subseteq (\bigcup_{l=1}^{M} \text{Recov}(l))$;
5 **if** *safety* $\wedge$ *recoverability* **then**
6 | Continue with AC;
7 **else**
8 | Switch to BC;
9 **end**

need to be over-approximated as convex sets to enable set intersection. When *safety* and *recoverability* evaluate to True, the plant is guaranteed to be i) safe in $[T, T + \Delta t]$ and ii) recoverable at $T + \Delta t$.

Next, we sketch a proof of the safety of P under the switching logic.

Lemma 1. *P remains safe during time $t \in [T, T + \Delta t]$, i.e. $\forall t \in [T, T + \Delta t]$: $\mathbf{x}_{AC}(t) \notin X_u$.*

Proof. The proof is based on the observation that after obtaining the sample $(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T))$, the switching logic ensures that AC does not drive P into unsafe states: if $Reach_{\leq \Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T))$ has a non-zero intersection with the unsafe states, then in step 3, *safety* will become False, resulting in a failover being performed in step 8. Note that $\mathbf{u}_{AC}(T)$ does not change in $[T, T + \Delta t]$ under Assumption 2. □

Lemma 2. *Every state sample $\mathbf{x}_{AC}(T)$ seen by the switching logic in step 1 of the algorithm is recoverable.*

Proof. The proof is based on induction. As the base case, we know that the initial states are recoverable. Consider the sample $\mathbf{x}_{AC}(T)$ and assume that it is recoverable. The set of all possible samples $\mathbf{x}_{AC}(T + \Delta t)$ is contained in $Reach_{=\Delta t}(\mathbf{x}_{AC}(T), \mathbf{u}_{AC}(T))$. If any of these reachable states lies outside the union of the *Recov(.)* sets of all of the modes of H_B, then it may be potentially non-recoverable. If such a state is reachable, then a failover will be triggered as *recoverability* will become False in step 4. Thus every state sample $\mathbf{x}_{AC}(T)$ seen in step 1 will be recoverable. □

Theorem 1. *The switching logic of the Simplex architecture defined in Algorithm 1 keeps P invariably safe.*

Proof. At every time step T, the switching logic ensures that P remains safe over the finite time horizon of length Δt as per Lemma 1. Additionally, it follows from Lemma 2 that the switching logic ensures that the next state sample $\mathbf{x}_{AC}(T+\Delta t)$ remains recoverable, and therefore safe. If the next state sample is potentially non-recoverable, indicated by *recoverability* becoming False in step 4, then the failover that is executed in step 8 ensures the plant remains safe under BC.

Thus, the switching logic ensures that P remains invariably safe. □

Algorithm 1 combines the offline computation of the BaCs with the online reachability computation at step 2 to guarantee the safety of the plant. The set intersection and the union operations involved in computing *safety* and *recoverability* in steps 2 and 3 must also be performed online, and add to the computational cost of performing on-the-fly reachability analysis.

Online reachability computations may be avoided by conservatively precomputing the sets for different partitions of the state space. The state space may be partitioned into different equivalence classes that reach the same sets of states in time up to, and at $T + \Delta t$ for conservative assumptions of the AC's inputs. Despite being computationally efficient, such a switching logic is prone to being overly conservative by not allowing the AC to operate over the largest possible region in the state space.

The switching logic in Algorithm 1 ensures that the operating region of the AC is maximized if the computation of the reachable sets is exact and the recoverable regions of the modes, obtained by intersecting the interior of the zero-level sets with the mode invariants, is maximal. This is often desired as the AC is intended to deliver better performance and/or serve mission-critical purposes.

4 Computing BaCs for Hybrid Systems

Let the hybrid system H and the descriptions of all the sets $I(l)$, $D(l)$, $Init(l)$, $Unsafe(l)$, $Guard(l', l)$, and $Reset(l', l)(x')$ be given along with some nonnegative constants $\sigma_{l',l}$, for each $l \in L$ and $(l, l') \in L^2, l \neq l'$. The search for a BaC for H can be cast as an instance of SOS optimization as follows. Find values of the coefficients which make the expressions

$$- B_l(x) - \sigma^T_{Init(l)}(x) g_{Init(l)}(x) \tag{5}$$

$$B_l(x) - \epsilon - \sigma^T_{Unsafe(l)}(x) g_{Unsafe(l)}(x) \tag{6}$$

$$- \frac{\partial B_l}{\partial x}(x) f_l(x, d) - \sigma_I(l)^T(x, d) g_{I(l)}(x) - \sigma^T_{D(l)}(x, d) g_{D(l)}(d) \tag{7}$$

$$- B_l(x) + \sigma_{l,l'} B_{l'}(x') - \sigma^T_{Guard(l,l')}(x, x') g_{Guard(l,l')}(x -$$

$$\sigma^T_{Reset(l,l')}(x, x') g_{Reset(l,l')}(x, x') \tag{8}$$

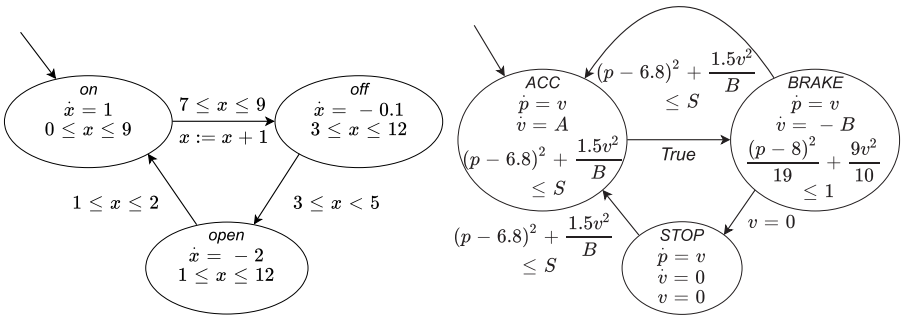
and the entries of $\sigma_{Init(l)}, \sigma_{Unsafe(l)}, \sigma_I(l)\sigma_{D(l)}, \sigma_{Guard(l,l')}, \sigma_{Reset(l,l')}$ sum of squares, for each $l \in L$ and $(l, l') \in L^2, l \neq l'$. See [16] for further details on computing BaCs for hybrid systems, e.g., the definitions of $g_{Init(l)}$ and $g_{Unsafe(l)}$, etc. Such SOS optimization programs can be solved using SOSTOOLS [16,17]. SOS optimization and SOSTOOLS itself have been applied to large systems, e.g., an industry-level hybrid system with 10-dimensional state in [8].

SOSTOOLS may run into numerical issues or provide incorrect solutions for some inputs as reported in [12,13]. We can overcome these issues by validating the BaCs using satisfiability modulo theory solvers, like Z3. Validation entails casting the negation of the assertion: for all relevant states, Eqs. 5–8 are non-negative. That is, Z3 looks for a state that makes any of these equations

negative. The domain of SMT formulae depends upon the mode or mode-pair under consideration. SOSTOOLS is re-parameterized if Z3 reports unsoundness of the solutions.

5 Case Studies

5.1 Case Study 1: Simple Water Tank System



(a) Water Tank System. (b) Stop Sign Obeying Controller.

Fig. 2. Hybrid automata of the composition H_B for the two case studies.

We consider a simple water tank system adopted from [15], where a controller seeks to keep the water level x in a tank between a certain range. Fig. 2a shows the hybrid automaton of the composition H_B. In mode *on*, the water tank is filled by a pump that increases the water level ($x' = 1$). The pump can be turned off when $x \ge 7$, and must be turned off when $x > 9$. More water pours in ($x := x + 1$) when the pump is shutting down.

In mode *off*, the pump is off and the valve is closed, but water leaks slowly ($x' = -0.1$). We assume that the valve must be opened completely (mode *open*) before reactivating the pump. The valve can be turned on when $x < 5$, and must be turned on when $x < 3$. In mode *open*, water drains quickly, and the system closes the valve and turns on the pump when $1 \le x \le 2$.

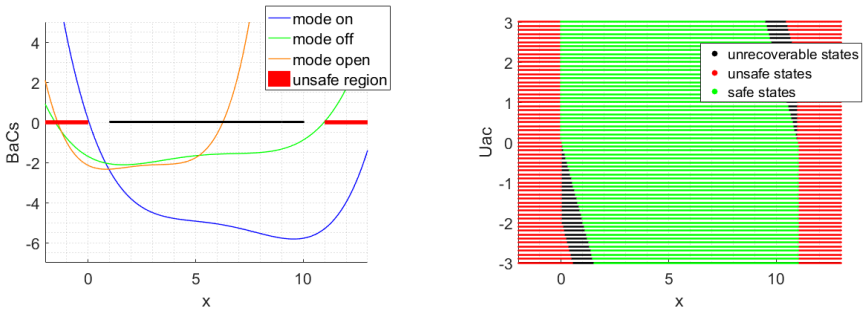
We assume that the disturbance in the continuous evolution is 0. The system is not asymptotically stable as the value of x varies within a certain range without reaching an equilibrium point. Its behavior is also nondeterministic.

Due to the fact that SOSTOOLS requires the minimum and maximum degrees of an SOS to be even, we made several changes to the original model of [15]. In particular, we modified the degrees of the invariants and guards to meet this requirement without affecting the behavior of the system.

BaCs and Switching Logic. Note that the zero-level sets of BaCs separate an unsafe region from all system trajectories. Thus, we need to have margins

between the unsafe region and the system trajectories. The unsafe region of the system is $X_u = \{x|x \leq 0\} \cup \{x|x \geq 11\}$. We compute the barrier certificates for the initial states $x_0 \in [1, 9]$. Since in each location l, x can only take a value within the invariant $I(l)$ of l, BaCs only need to satisfy Eqs. 5- 8 in the invariant set $I(l)$ [16]. We take the intersection of X_u and $I(l)$ as the unsafe region for mode l. The runtimes needed to compute the BaCs using SOSTOOLS and to validate them on Z3 are 1.142s and 0.206s, respectively, on an Intel Core i7-4770 CPU @ 3.4 GHz with 16GB RAM.

Fig. 3a shows the resulting BaCs. The zero-level set of the BaCs for modes *on*, *off*, and *open* is $\{x = 0.0384\}$, $\{x = 10.9709\}$, and $\{x = 6.2624\}$, respectively. Note that we only consider the zero-level set within the invariant of the mode. The recoverable regions are $\text{Recov}(on) = [0.0384, 9]$, $\text{Recov}(off) = [3, 10.9709]$, and $\text{Recov}(open) = [1, 6.2624]$. The figure shows that the intersection of the interior of the zero-level set and the invariant of each mode separates the unsafe region from all system trajectories. The validation of the BaCs using Z3 proves that they satisfy all of the conditions. We obtain the recoverable region of the system as $\bigcup_{l=1}^{M} \text{Recov}(l) = [0.0384, 10.9709]$.



(a) BaCs of the modes. The red lines represent the unsafe regions. The black line shows the system trajectory.

(b) Safe, unsafe and unrecoverable states with $\Delta t = 0.5$ under different control inputs $\mathbf{u}_{AC}$.

Fig. 3. BaCs of the system and snapshot of the switching logic at run-time.

As an instance of the Simplex architecture, the water tank system could be controlled by an advanced controller with a more complex control objective. At any given time, the decision module in the Simplex architecture decides whether or not to switch to BC based on Algorithm 1. Fig. 3b illustrates a snap-shot of the switching logic at time T across the state space. For illustration purposes, we discretize the state space and apply the switching logic for each discrete state with the corresponding control input $\mathbf{u}_{AC}$. Note that the control input is applied to the water level, i.e., $\dot{x} = \mathbf{u}_{AC}$. We have $x(T + \Delta t) = x(T) + \mathbf{u}_{AC} \cdot \Delta t$.

To check the *safety* condition in Algorithm 1, it is sufficient to check if the intersection of the line segment from $x(T)$ to $x(T + \Delta t)$ and the unsafe region

X_u is empty. A red dot is used in Fig. 3b to represent unsafe states that do not satisfy the *safety* condition. To check the *recoverability* condition, we use the recoverable region computed above and check if $x(T + \Delta t) \in \bigcup_{l=1}^{M} \text{Recov}(l)$. A black dot is used to represent the unrecoverable states that do not satisfy the *recoverability* condition. We switch to BC if the current state is unsafe or unrecoverable. If the current state satisfies both *safety* and *recoverability*, shown as a green dot, we continue with the AC. Note that if $\mathbf{u}_{AC} < 0$, then the smaller $\mathbf{u}_{AC}$ is, the larger lower bound we have for the safe states. Also, when $\mathbf{u}_{AC} > 0$, the larger $\mathbf{u}_{AC}$ is, the smaller upper bound we have for the safe states.

5.2 Case Study 2: Stop-Sign-Obeying Controller

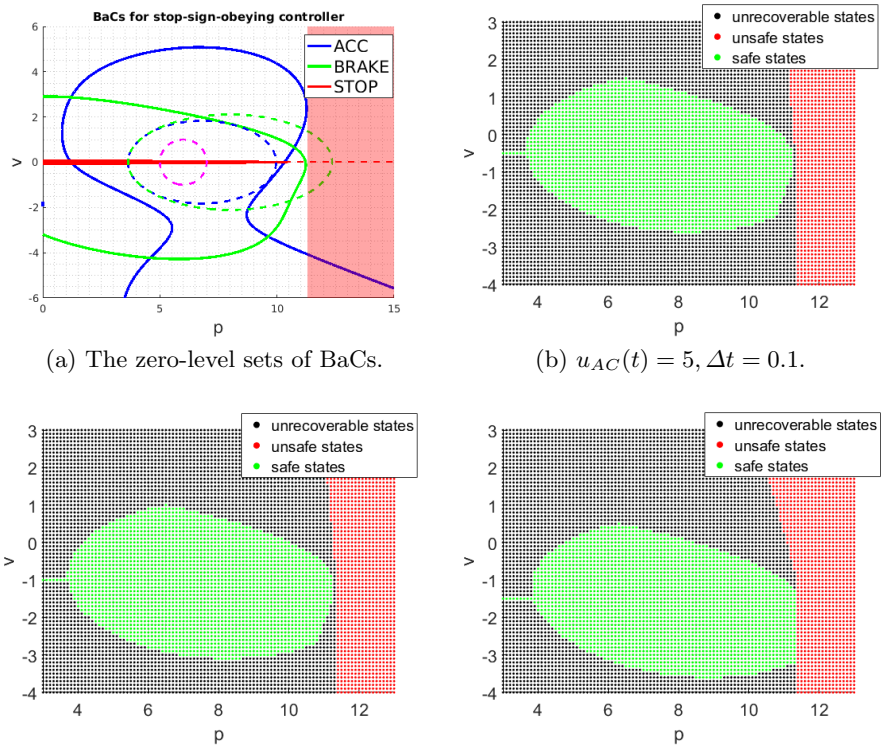
Our second case study is a stop-sign-obeying controller of a car that chooses when to begin decelerating so that it stops at or before a stop sign [18]. Fig. 2b shows the corresponding hybrid automaton H_B. The system of stop-sign obeying controller has 2 variables with second order derivatives and quadratic functions as guards and invariants.

The state variables p and v denote the position and velocity of the car, respectively and S denotes the position of the stop sign. In mode ACC, the car accelerates with a constant rate A. It can stay in the mode as long as the invariant is satisfied, or switch nondeterministically to mode $BRAKE$. In mode $BRAKE$, the car decelerates with a constant rate $-B$. It can switch nondeterministically to mode ACC if the guard condition is satisfied. It also switches to mode $STOP$ if $v = 0$. In mode $STOP$, the velocity and the acceleration are both 0. We assume that the disturbance in the continuous evolution is 0.

Note that, due to the practical limitation of SOSTOOLS, also mentioned in Section 5.1, we slightly modify some guards and invariants of stop-sign-obeying controller. The original model can be found in [18].

BaCs and Switching Logic. To compute BaCs, we consider ACC as the initial mode and $\{(p, v)|(p-6)^2 + v^2 \geq 1\}$ as the initial set. We choose $A = 1$, $B = 0.5$ and $S = 11.3$. To compute feasible BaCs, we consider the stop sign is at $p = S - d$ during the computation, where d is some non-negative offset. This allows us to maintain a safe margin ($d > 0$) between the unsafe region and all possible system trajectories. Using a simple binary search, we find $d = 1.3$ as the smallest possible value in the interval $[0, 2]$, a reasonable safety margin in this case. We set $\sigma_{l,l'} = 0$ for all $(l, l') \in L^2$ in Eq. 8.We validate the resulting BaCs on Z3 as discussed in Section 4. The runtimes to compute the BaCs using SOSTOOLS and to validate them on Z3 are 1.497s and 0.295s, respectively, on an Intel Core i7-4770 CPU @ 3.4 GHz with 16GB RAM.

In Fig. 4a, the solid curves represent zero-level sets of the computed BaCs, whereas the regions with dashed boundary are the corresponding mode invariants. The dashed red ellipse and pink rectangular region represent the initial and unsafe states, respectively. The figure shows that the intersection of the interior of zero-level set of the BaC for any mode and its mode invariant does not intersect with the unsafe region. This ensures that the union of all the intersections of



(a) The zero-level sets of BaCs. (b) $u_{AC}(t) = 5, \Delta t = 0.1$.

(c) $u_{AC}(t) = 10, \Delta t = 0.1$. (d) $u_{AC}(t) = 5, \Delta t = 0.5$.

Fig. 4. (a) Illustration of BaCs. (b)-(d) Online switching decision at time t in stop-sign-obeying controller across the state-space for a given $\mathbf{u}_{AC}(t)$ of AC and Δt.

interior of zero-level set of BaC with its corresponding mode invariant, denoted as $\bigcup_{l=1}^{M} \text{Recov}(l)$, can be used as the recoverable sets in the switching logic.

Fig. 4 illustrates a snap-shot of switching logic at run-time t across the state-space in stop-sign-obeying controller in three different cases. For the illustration purpose, we consider a discrete state-space. For each discrete state, we apply the decision logic by computing both $Reach_{\leq \Delta t}(\mathbf{x}_{AC}(t), \mathbf{u}_{AC}(t))$ and $Reach_{=\Delta t}(\mathbf{x}_{AC}(t), \mathbf{u}_{AC}(t))$, where $\mathbf{x}_{AC} = [p, v]^T$ and $\mathbf{u}_{AC}(t)$ is some acceleration input provided by AC at t, i.e, $v' = \mathbf{u}_{AC}(t)$. For $\mathbf{x}(t+\Delta t) = [p(t+\Delta t), v(t+\Delta t)]^T$, we have $v(t+\Delta t) = v(t)+\mathbf{u}_{AC}(t)\Delta t$ and $p(t+\Delta t) = p(t)+v(t)\Delta t+\frac{1}{2}\mathbf{u}_{AC}(t)\Delta t^2$. To check the *safety* condition in Algorithm 1, we check if the intersection of the enclosure from $\mathbf{x}(t)$ to $\mathbf{x}(t + \Delta t)$ (which can be over-approximated with a rectangle) and the unsafe region X_u is empty. To check the *recoverability* condition, we use the recoverable region computed above and check if $\mathbf{x}(t + \Delta t) \in \bigcup_{l=1}^{M} \text{Recov}(l)$. If $\mathbf{x}_{AC}(t)$ satisfies both *safety* and *recoverability* condition of Algorithm 1, we call it a safe state (green dot) and continue with AC. But if it only

satisfies the *safety* condition, we call the state as unrecoverable (black dot) and switch to BC. The states represented as red dots do not satisfy any condition, i.e., it is either unsafe or will go to unsafe region within Δt. Among the subfigures of Fig. 4, we either vary $\mathbf{u}_{AC}(t)$ or Δt. Note that we assume some arbitrary values for $\mathbf{u}_{AC}(t)$ for all cases that AC may provide during run-time.

6 Related Work

Our BaC-based approach is similar to the combination of offline and online strategies in [9], but there are some key differences. The authors use the following switching logic. The AC controls the plant if it is well within the largest ellipsoidal safe sublevel set of the Lyapunov function that establishes the safety of the BC. If the plant is near the boundary of the ellipsoid, the AC retains control if reachability analysis shows that i) the plant will remain safe under the control of the BC over a finite horizon and ii) the BC can guarantee to bring the plant back into the ellipsoid, thus guaranteeing recoverability, at the end of the finite horizon. The logic is designed to maximize the AC's operating region. Moreover, the plant is allowed to leave the ellipsoid as long as it is guaranteed to be recoverable at the end of the finite horizon. Our Algorithm 1 also achieves this objective. The recoverability test in step 4 checks if all the states reachable under the AC's control at the end of Δt are recoverable. In other words, the plant is allowed to cross the zero-level sets of the barrier certificates if it is guaranteed to return into at least one of the zero-level sets at the end of Δt.

Additionally, [9] relies on LMI, which is primarily intended for stability analysis of linear systems; nonlinear systems must be linearized for analysis. BaCs, on the other hand, inherently encode the notion of safety for hybrid automata and other nonlinear systems. Our approach enables us to go beyond simple single-mode systems, like the inverted pendulum model of [9], and design Simplex architectures for multi-mode hybrid systems. Specifically, we detail the relationship between the reach-sets and the BaC-based recoverable regions.

We also make a simplifying assumption: the decision module can observe the control input produced by the AC, and that the control input does not change during the decision period, which is the same as the control period. This assumption eliminates the need to abstract AC as a hybrid automaton.

The concept of Simplex is closely related to Run-Time Assurance (RTA). BaCs were proposed for RTA of hybrid systems in [11], but the switching logic was not described in detail. Moreover, the details of computing BaCs and case studies were not presented. In [3], reachability analysis on hybrid systems is applied to produce a decision module that guarantees safety, which is completely offline with assumptions about the maximum derivative of the states. By contrast, our online computation assumes that the current control input is known.

In [22], compositional barrier functions are used to guarantee the simultaneous satisfaction of composed objectives. They rely on a single controller and an optimization-based approach to correct the controller in a minimally invasive

fashion when violations of safety are imminent. This approach is limited by the single controller, and consequently less flexible compared to Simplex.

7 Conclusions and Future Work

We presented a Barrier-Certificates-based two-controller Simple Architecture for hybrid systems. In addition to establishing safety of the plant under the baseline controller, the zero-level sets of the BaCs also yield recoverable regions, where the safety is guaranteed for infinite time. The switching logic of the architecture, which samples the state of the plant under the advanced controller periodically in discrete time, uses on-the-fly reachability computations to ensure that i) the plant remains safe between successive samples and ii) every sample is recoverable. Two case studies, a water-tank system and a stop-sign-obeying controller, were presented to illustrate the implementation aspects of our approach.

We plan to extend our work along several directions. We will pursue the computation of barrier certificates that guarantee optimal switching, which ensures that the operating region of the advanced controller is maximized. Our approach will be applied to more complicated systems with nonlinearities and exogenous inputs. Finally, we will extend our approach to compositions of barrier certificates that simultaneously satisfy multiple composed safety constraints.

References

1. Asarin, E., Dang, T., Girard, A.: Reachability analysis of nonlinear systems using conservative approximation. In: Proceedings of the 6th International Workshop on Hybrid Systems: Computation and Control (HSCC), pp. 20–35. Springer (2003)
2. Bak, S.: Verifiable COTS-Based Cyber-Physical Systems. Ph.D. thesis, University of Illinois at Urbana-Champaign (2013)
3. Bak, S., Manamcheri, K., Mitra, S., Caccamo, M.: Sandboxing controllers for cyber-physical systems. In: Proceedings of the 2011 IEEE/ACM International Conference on Cyber-Physical Systems ICCPS. pp. 3–12. IEEE Computer Society (2011)
4. Boyd, S.P., El Ghaoui, L., Feron, E., Balakrishnan, V.: Linear matrix inequalities in system and control theory, vol. 15. SIAM (1994)
5. Chen, X., Sankaranarayanan, S.: Decomposed reachability analysis for nonlinear systems. In: Real-Time Systems Symposium (RTSS). pp. 13–24. IEEE (2016)
6. Chutinan, A., Krogh, B.H.: Computational techniques for hybrid system verification. IEEE Transactions on Automatic Control 48(1), 64–75 (2003)
7. Dang, T., Maler, O.: Reachability analysis via face lifting. In: Proceedings of the First International Workshop on Hybrid Systems: Computation and Control (HSCC). pp. 96–109. Springer (1998)
8. Glavaski, S., Papachristodoulou, A., Ariyur, K.: Safety Verification of Controlled Advanced Life Support System Using Barrier Certificates, pp. 306–321. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
9. Johnson, T.T., Bak, S., Caccamo, M., Sha, L.: Real-time reachability for verified simplex design. ACM Transactions on Embedded Computing Systems (TECS) 15(2), 26 (2016)
10. Loechner, V.: Polylib: A library for manipulating parameterized polyhedra (1999)

11. Murthy, A., Bartocci, E., Zadok, E., Stoller, S., Smolka, S., Grosu, R.: Simplex architecture for run time assurance of hybrid systems. In: Safe and Secure Systems and Software Symposium (S5) (2012)
12. Murthy, A., Islam, M.A., Smolka, S.A., Grosu, R.: Computing bisimulation functions using SOS optimization and δ-decidability over the reals. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control. pp. 78–87. ACM, New York, NY, USA (2015)
13. Murthy, A., Islam, M.A., Smolka, S.A., Grosu, R.: Computing compositional proofs of input-to-output stability using sos optimization and δ-decidability. Nonlinear Analysis: Hybrid Systems 23, 272–286 (2017)
14. Papachristodoulou, A., Anderson, J., Valmorbida, G., Prajna, S., Seiler, P., Parrilo, P.A.: SOSTOOLS: Sum of squares optimization toolbox for MATLAB (2013)
15. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer Berlin Heidelberg (2010)
16. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: In Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC). pp. 477–492. Springer (2004)
17. Prajna, S., Jadbabaie, A., Pappas, G.J.: A framework for worst-case and stochastic safety verification using barrier certificates. IEEE Transactions on Automatic Control 52(8), 1415–1429 (2007)
18. Quesel, J.D., Mitsch, S., Loos, S., Aréchiga, N., Platzer, A.: How to model and prove hybrid systems with keymaera: A tutorial on safety. International Journal on Software Tools for Technology Transfer 18(1) (2016)
19. Seto, D., Sha, L.: An engineering method for safety region development. Technical Report CMU/SEI-99-TR-018, Software Engineering Institute (1999)
20. Sha, L.: Using simplicity to control complexity. IEEE Software 18(4), 20–28 (2001)
21. Tomlin, C.J., Mitchell, I., Bayen, A.M., Oishi, M.: Computational techniques for the verification of hybrid systems. Proceedings of the IEEE 91(7), 986–1001 (2003)
22. Wang, L., Ames, A.D., Egerstedt, M.: Multi-objective compositions for collision-free connectivity maintenance in teams of mobile robots. CoRR abs/1608.06887 (2016)