Eliminating Spurious Transitions in Reachability with Support Functions

Goran Frehse
Université Grenoble 1 Joseph Fourier - Verimag
Centre Equation, 2 av. de Vignate
38610 Gières, France
goran.frehse@imag.fr

Sergiy Bogomolov
University of Freiburg
Georges-Köhler-Allee 52
79110 Freiburg, Germany
bogom@informatik.uni-freiburg.de

Marius Greitschus
University of Freiburg
Georges-Köhler-Allee 52
79110 Freiburg, Germany
greitsch@informatik.uni-freiburg.de

Thomas Strump
University of Freiburg
Georges-Köhler-Allee 52
79110 Freiburg, Germany
strumpt@informatik.uni-freiburg.de

Andreas Podelski
University of Freiburg
Georges-Köhler-Allee 52
79110 Freiburg, Germany
podelski@informatik.uni-freiburg.de

ABSTRACT

Computing an approximation of the reachable states of a hybrid system is a challenge, mainly because representing the solutions of ODEs with a finite number of sets does not scale well. Using template polyhedra to cover the solution greatly reduces the computational complexity, since it replaces complex operations on sets by a small number number of optimization problems. However, the use of templates may make the overapproximation too conservative. Spurious transitions (which are falsely considered reachable) are particularly detrimental to performance and accuracy, and may exacerbate the state explosion problem. In this paper, we examine how spurious transitions can be avoided with minimal computational effort. To this end, detecting spurious transitions is reduced to the well-known problem of showing that two convex sets are disjoint by finding a hyperplane that separates them. We generalize this to flowpipes by considering hyperplanes that evolve with time in correspondence to the dynamics of the system. The approach is implemented in the model checker SpaceEx and demonstrated on examples.

Categories and Subject Descriptors

G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations—*Initial value problems*

Keywords

Hybrid systems, verification, reachability, tools

1. INTRODUCTION

A major bottleneck in computing the reachable states of a hybrid automaton is the approximation of the states reachable by time elapse, i.e., approximating all solutions of the ODEs with suitable sets. We call this *flowpipe approximation.* Support functions lead to a scalable algorithm that can be arbitrarily precise [8], and similar techniques can be applied using template polyhedra [11, 3]. The approximation error depends on the time step and the directions in which the support function is evaluated.

In our experiences with applying scalable flowpipe approximation algorithms, the number of continuous sets that are produced tends to grow quickly and become a limiting factor. Clustering is usually applied to help reduce this number, and optimal clustering can be carried out with support functions [5]. This number of sets is aggravated dramatically by spurious transitions, i.e., transitions that are enabled as an artifact of the overapproximation. The approximation accuracy can be improved by reducing time steps and increasing the number of directions, but if done indiscriminately this leads to large computational cost: to guarantee a Hausdorff error of ε in n dimensions, the support function must be evaluated $O(1/\varepsilon^{n-1})$ times [9].

We propose a procedure to show that a transition is spurious, i.e., its guard set is unreachable. It aims at using as few directions as possible, and adjusting the accuracy automatically. We call this separating the guard set from the flowpipe (as opposed to safety), in order to differentiate it from showing safety over all runs of the hybrid automaton. Our approach is based on the separation of two convex sets: efficient algorithms are known that produce a hyperplane separating the two sets, and its normal vector is a suitable template direction for the support function algorithm.

We propose two different ways to turn the flowpipe separation problem into a sequence of convex separation problems. In a *convexification-based* approach, we approximate the flowpipe with a finite number of convex set as in [5]. To each of these sets, we apply the above convex separation algorithm. In a *point-wise* approach, we run the convex separation algorithm at discrete points in time. The result (separation or overlap) is propagated along the time axis using continuous-time bounds on the support function of the flowpipe computed as in [5]. The main contributions of the paper are as follows:

- We propose a novel construction of inner approximations of convex sets based solely on support functions (not support vectors). This construction is sound even for approximate computations. (Sect. 2.2)
- We propose a novel procedure for separating convex sets using only approximately computed values of support functions. To the best of our knowledge, this is the first such procedure using only support functions (not support vectors) and the first that is sound even for approximate computations. (Sect. 4.1)
- For the *point-wise* approach, we incorporate both *static directions*, where we check for how long the same hyperplane (possibly shifted) still separates the flowpipe (Sect. 5.2.1), and *dynamic directions*, where we rotate the separating hyperplane with the adjunct dynamics of the system (Sect. 5.2.2). These methods are complimentary since there are systems where either one or the other technique, but not both, can show separation over an infinite time horizon.

The problem of showing that a given "unsafe" set (in our case, the guard set) is not reachable is known as the safety problem. Various approaches exist, and due to lack of space we cite only a small selection. In [2], predicate abstractions are used to refute counter-examples of hybrid systems. The separating hyperplanes that we construct can be viewed as such predicates, although in our setting they need only be satisfied over intervals of time. In [4], abstractions based on eigenforms are refined using counter examples until safety is shown. However the approach is limited to deterministic dynamics, while we can handle additive nondeterminism in the ODEs. Alternating forward and backward reachability between the initial and the unsafe set can be used to show safety, but there are inherent problems with numerical accuracy, since a stable system becomes unstable when going backwards in time [10]. The main difference to all these approaches is that we are only looking for a technique to detect as quickly as possible when a set is unreachable within a location; the goal is not to decide the safety problem.

The remainder of the paper is organized as follows. In the next section, we present approximate support functions, which we use to represent convex sets that can be only computed approximately. In Sect. 3, we briefly recall the flowpipe approximation from [5], which uses approximate support functions, and relate spurious transitions to flowpipe separation. In Sect. 4, we present our algorithms for approximating and separating convex sets based on approximately computed support functions. These algorithms are applied to flowpipe separation using convexification in Sect. 5.1, and using point-wise separation in Sect. 5.2. Experimental results are shown in Sect. 6.

Note to reviewers: For lack of space, the proofs omitted from this paper are available for anonymous download at `http://www-verimag.imag.fr/~frehse/hscc15_proofs.pdf`

2. REPRESENTING SETS WITH APPROXIMATE SUPPORT FUNCTIONS

A convex set can be represented by its support function, which attributes to each direction in $\mathbb{R}^n$ the signed distance of the farthest point of the set to the origin. Computing the value of the support function for a given set of directions, one obtains a polyhedron that overapproximates the set. In this paper, we consider this computation to be approximative, i.e., only a lower and an upper bound on the support function can be computed.

We recall some basics. A *halfspace* $\mathcal{H} \subseteq \mathbb{R}^n$ is the set of points satisfying a linear constraint, $\mathcal{H} = \{x \mid a^{\mathsf{T}}x \le b\}$, where $a = (a_1 \cdots a_n) \in \mathbb{R}^n$ and $b \in \mathbb{R}$. A *polyhedron* $\mathcal{P} \subseteq \mathbb{R}^n$ is the intersection of a finite number of halfspaces

$$\mathcal{P} = \Big\{x \;\Big|\; \bigwedge\nolimits_{i=1}^{m} a_i^{\mathsf{T}}x \le b_i\Big\},$$

where $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$. A *polytope* is a bounded polyhedron. The *convex hull* $\mathrm{CH}(\mathcal{X}) \subseteq \mathbb{R}^n$ of a set $\mathcal{X}$ is

$$\mathrm{CH}(\mathcal{X}) = \Big\{\sum_{i=1}^{m} \lambda_i v_i \;\Big|\; v_i \in X, \lambda_i \in \mathbb{R}^{\ge 0}, \sum_{i=1}^{m} \lambda_i = 1\Big\}.$$

The *support function* of a closed and bounded continuous set $\mathcal{X} \subseteq \mathbb{R}^n$ with respect to a direction vector $\ell \in \mathbb{R}^n$ is

$$\rho_{\mathcal{X}}(\ell) = \max\{\ell^{\mathsf{T}}x \mid x \in \mathcal{X}\}.$$

The set of *support vectors* (or *maximizers*) of $\mathcal{X}$ in direction ℓ is denoted by

$$\sigma_{\mathcal{X}}(\ell) = \{x^* \in \mathcal{X} \mid \ell^{\mathsf{T}}x^* = \rho_{\mathcal{X}}(\ell)\}.$$

2.1 Approximate Support Functions

An *approximate support function* is a function $\mathsf{support}$ that given a direction ℓ and an accuracy $\varepsilon > 0$ produces an upper bound on the support function. We require the bound to be within ε of the true value:

$$\mathsf{support}(\mathcal{X}, \ell, \varepsilon) - \varepsilon \le \rho_{\mathcal{X}}(\ell) \le \mathsf{support}(\mathcal{X}, \ell, \varepsilon). \quad (1)$$

Computing an approximate support function for a given set of directions provides an outer and an inner approximation of the set. Consider a set of directions $L = \{\ell_1, \dots, \ell_N\}$ and values $s_k^+ = \mathsf{support}(\mathcal{X}, \ell_k, \varepsilon)$ for $i = 1, \dots, N$. This gives the *outer approximation*

$$\lceil \mathcal{X} \rceil_L = \bigcap_{k=1,\dots,K} \{\ell_k^{\mathsf{T}}x \le s_k^+\}, \quad (2)$$

which satisfies $\mathcal{X} \subseteq \lceil \mathcal{X} \rceil_L$. As shown in Fig. 1, at least one point $x \in \mathcal{X}$ is inside the *facet slab* associated with ℓ_k,

$$\lfloor \mathcal{X} \rfloor_k = \lceil \mathcal{X} \rceil_L \cap \{\ell_k^{\mathsf{T}}x \ge s_k^+ - \varepsilon\}. \quad (3)$$

Using these constructs, we have the following bounds on the support function of $\mathcal{X}$.

Lemma 2.1. *[5] Given a set of directions $L = \{\ell_1, \dots, \ell_N\}$ and values $s_k^+ = \mathsf{support}(\mathcal{X}, \ell_k, \varepsilon)$ for $i = 1, \dots, N$, the support function is bounded by $\rho_{\mathcal{X}}^-(\ell) \le \rho_{\mathcal{X}}(\ell) \le \rho_{\mathcal{X}}^+(\ell)$, where*

$$\rho_{\mathcal{X}}^+(\ell) = \rho_{\lceil \mathcal{X} \rceil}(\ell), \quad (4)$$

$$\rho_{\mathcal{X}}^-(\ell) = \max_{k=1,\dots,N} -\rho_{\lfloor \mathcal{X} \rfloor_k}(-\ell). \quad (5)$$

The above bounds on the support function can be used to compute an inner approximation, i.e., a set of points that are guaranteed to be inside the set. This will be discussed next.

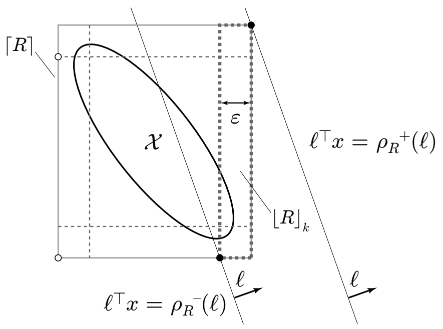


Figure 1: The outer approximation $\lceil \mathcal{X} \rceil$ (solid grey) for a convex set $\mathcal{X}$ in the positive and negative axis directions. The facet slab $\lfloor \mathcal{X} \rfloor_k$ (dashed), contains at least one point of $\mathcal{X}$. The support function of $\mathcal{X}$ is is bounded by the interval $[\rho_R^-(\ell), \rho_R^+(\ell)]$

2.2 Inner Approximation of Approximate Support Functions

Once the support function of a set $\mathcal{X}$ has been evaluated a number of times, the obtained values can be used to construct facet slabs. From these facet slabs we can derive an underapproximation of $\mathcal{X}$ by using the following criterion: a point x is in $\mathcal{X}$ if

$$\forall x_1 \in \lfloor \mathcal{X} \rfloor_1, \ldots, x_N \in \lfloor \mathcal{X} \rfloor_N : x \in \mathrm{CH}(x_1, \ldots, x_N). \qquad (6)$$

However, this set is costly to compute because it involves quantifier alternation and bilinear constraints.

To obtain an underapproximation of $\mathcal{X}$ with (relatively) little cost, we first estimate a point for each facet, then construct their convex hull, and finally shrink this set sufficiently to be sure that it is an underapproximation.

To estimate a point for a facet of the approximation, we use a center. A point $x \in \mathcal{X}$ is a *Chebyshev center* of $\mathcal{X}$ if it is the center of the largest ball that lies inside $\mathcal{X}$. The set of Chebyshev centers satisfies all constraints when they are tightened by the same amount, and as long as they are satisfiable. If the polyhedron is flat, i.e., its constraints contain equalities, these centers degenerate. We therefore compute them regarding the relative interior of $\mathcal{P}$. Assuming $\mathcal{P}$ has k equalities, let $\mathcal{P} = \{\bigwedge_{i=1}^k a_i^\mathsf{T} x = b_i \wedge \bigwedge_{i=k+1}^m a_i^\mathsf{T} x \le b_i\}$. The *relative Chebyshev center* x^* and its radius z^* are

$$< x^*, z^* >= \operatorname*{argmax}_{x, z \ge 0} z \text{ s.t. } \bigwedge\nolimits_{i=1}^{k} a_i^\mathsf{T} x = b_i \wedge \\ \bigwedge\nolimits_{i=k+1}^{m} a_i^\mathsf{T} x + \|a_i\| z \le b_i. \qquad (7)$$

Given any points $c_1, \ldots, c_N \in \lceil \mathcal{X} \rceil_L$, we define our underapproximation by shrinking their convex hull as follows.

Proposition 2.2. *Given a set of points $c_1, \ldots, c_N \in \lceil \mathcal{X} \rceil_L$, let a_i be the normal vectors of their convex hull, i.e.,*

$$\mathrm{CH}(c_1, \ldots, c_N) = \{\bigwedge\nolimits_{i=1}^{M} a_i^\mathsf{T} x \le b_i\}.$$

Let J_i be the indices of the points that lie on the border of the i-th constraint, i.e., $J_i = \{j \mid a_i^\mathsf{T} c_j = b_i\}$, and let

$$b_i^- = \min_{j \in J_i} -\rho_{\lfloor \mathcal{X} \rfloor_j}(-a_i).$$

Then the set $\mathcal{C}^- = \{c \mid \bigwedge_{i=1}^M a_i^\mathsf{T} c \le b_i^-\}$ is a subset of $\mathcal{X}$.

3. REACHABILITY WITH SUPPORT FUNCTIONS

We consider hybrid systems modeled by a *hybrid automaton*. An approximation of its reachable states can be obtained by computing successor states with respect to time elapse and discrete transitions (jumps), and repeating the process until all of the successors states have been encountered in a previous step. This procedure need not terminate, and the problem is undecidable in general. Since the details of the reachability algorithm have been reported elsewhere and are not essential for the results of this paper, we provide a brief summary and refer the reader to [6].

In the the following section, we define the class of hybrid automata we consider in this paper. We then recall the scalable flowpipe approximation algorithm from [5], which is extensively used in the remainder of the paper. The image computation with respect to a discrete transitions is presented since it relates eliminating spurious transitions to the problem of separating the flowpipe from a convex set.

3.1 Hybrid Automata

A *hybrid automaton* $H = (\mathit{Loc}, \mathit{Inv}, \mathit{Flow}, \mathit{Trans}, \mathit{Init})$ is defined as follows [1]. It has a set of discrete states Loc called *locations*. Each $l \in \mathit{Loc}$ is associated with a set of differential equations (or inclusions) $\mathit{Flow}(l)$ that defines the time-driven evolution of the continuous variables. A *state* $s \in \mathit{Loc} \times \mathbb{R}^n$ consists of a location and values for the n continuous variables. A set of *discrete transitions* Trans defines how the state can jump between locations and instantaneously modify the values of continuous variables. A jump can take place when the state is inside the transition's *guard* set, and the target states are given by the transition's *assignment*. The system can remain in a location l while the state is inside the *invariant* set $\mathit{Inv}(l)$. All behavior originates from the set of *initial states* Init.

In this paper, we consider $\mathit{Flow}(l)$ to be continuous dynamics of the form

$$\dot{x}(t) = Ax(t) + u(t), \qquad u(t) \in \mathcal{U}, \qquad (8)$$

where $x(t) \in \mathbb{R}^n$ is an n-dimensional vector, $A \in \mathbb{R}^n \times \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^n$ is a closed and bounded convex set. Transition assignments are of the (deterministic) affine form

$$x' = Rx + w, \qquad (9)$$

where $x' \in \mathbb{R}^m$ denotes the values after the transition, $R \in \mathbb{R}^m \times \mathbb{R}^n$ and $w \in \mathbb{R}^m$.

We compute the reachable states by recursively computing the image of the initial states with respect to time elapse and discrete transitions until a fixpoint is reached. Before we can discuss the image computation, we present how we describe sets of states with approximate support functions.

3.2 Flowpipe Approximation

In a given location of the hybrid automaton, we refer to the states reachable from an initial set $\mathcal{X}_0$ by time elapse as the *flowpipe* of $\mathcal{X}_0$. Given an initial set $\mathcal{X}_0$, the *reachable states at time t* is the set of values of the solutions of (8) with initial condition $x(0) \in \mathcal{X}_0$. We denote this set with

$$\mathrm{Reach}_t(\mathcal{X}_0, A, \mathcal{U}) = e^{At}\mathcal{X}_0 \oplus \int_0^t e^{As}\mathcal{U} ds. \qquad (10)$$

To simplify the notation, let $\mathcal{X}_t = \text{Reach}_t(\mathcal{X}_0, A, \mathcal{U})$. For affine dynamics, $\mathcal{X}_t$ is convex for any given t, so $\mathcal{X}_t$ can be represented by its support function. The flowpipe from the initial states $\mathcal{X}_0$ over the time interval $[t_1, t_2]$ is the set $\mathcal{X}_{t_1,t_2} = \bigcup_{t_1 \le t \le t_2} \mathcal{X}_t$.

We now summarize the flowpipe approximation algorithm in [5]. It is based on approximating the support function of $\mathcal{X}_t$ over time as follows. Given a time interval $[t_1, t_2]$, a direction d, and an accuracy bound $\varepsilon > 0$, it constructs a piecewise linear function

$$s^+_{d,\varepsilon}(t) = \text{sReach}(\mathcal{X}_0, A, \mathcal{U}, [t_1, t_2], d, \varepsilon)$$

such that for all $t \in [t_1, t_2]$,

$$s^+_{d,\varepsilon}(t) - \varepsilon \le \rho_{\mathcal{X}_t}(d) \le s^+_{d,\varepsilon}(t) \qquad (11)$$

Let $D = \{d_1, \ldots, d_m\}$ be a set of directions for which $s^+_{d_i,\varepsilon}(t)$ has been computed. This defines a flowpipe approximation pointwise in time,

$$\Omega_t = \bigcap_{d_i \in D} \{d_i^\mathsf{T} x \le s^+_{d_i,\varepsilon}(t)\}, \text{ with } \mathcal{X}_t \subseteq \Omega_t$$

so the union $\Omega_{t_1,t_2} = \bigcup_{t_1 \le t \le t_2} \Omega_t$ contains $\mathcal{X}_{t_1,t_2}$. It can be shown that Ω_{t_1,t_2} is always a finite union of convex polyhedra, $\Omega_{t_1,t_2} = \bigcup_{j=0}^{N} \Omega_j$. Each Ω_j approximates $\mathcal{X}_t$ over an interval of time $[t_j, t_{j+1}]$, and it is possible to construct the smallest number N of such sets for a given bound on the total approximation error.

Each Ω_j can be refined in the following sense: Given an additional direction d' and accuracy ε', one computes $s_{d',\varepsilon'}(t)$. Then either (a) it is possible to add (convex) constraints to Ω_j such that it reflects these bounds, i.e.,

$$\Omega'_j = \bigcup_{t_1 \le t \le t_2} \bigcap_{d_i \in D} \{d_i^\mathsf{T} x \le s^+_{d_i,\varepsilon}(t)\} \cup \{{d'}^\mathsf{T} x \le s^+_{d',\varepsilon'}(t)\},$$

or (b) Ω_j needs to be replaced by more than one convex set in order to maintain the desired accuracy.

3.3 Eliminating Spurious Transitions

Let $\mathcal{G}$ be the guard set of the transition, $\mathcal{I}^-$ the invariant of the source location, $\mathcal{I}^+$ the invariant of the target location, and let the transition assignment be (9). We assume $\mathcal{G}$,$\mathcal{I}^-$,$\mathcal{I}^+$ to be polyhedra and assume that the set of template directions L contains the normal vectors of the constraints of these polyhedra. Let the target invariant be

$$\mathcal{I}^+ = \left\{x \;\middle|\; \bigwedge_{i=1}^m \bar{a}_i^\mathsf{T} x \le \bar{b}_i\right\}.$$

The image of a set $\mathcal{X}$ with respect to the transition is

$$post_d(\mathcal{X}) = \big(R(\mathcal{X} \cap \mathcal{G} \cap \mathcal{I}^-) \oplus w\big) \cap \mathcal{I}^+. \qquad (12)$$

Let G^* be the intersection of the guard, the source invariant and the back-transformed target invariant,

$$\mathcal{G}^* = \mathcal{G} \cap I^- \cap \left\{x \;\middle|\; \bigwedge_{i=1}^m \bar{a}_i^\mathsf{T} R x \le \bar{b}_i - w^\mathsf{T} \bar{a}_i\right\}. \qquad (13)$$

Using $\mathcal{G}^*$, the image operator can be simplified so that it involves a single intersection operation [7]:

$$post_d(\mathcal{X}) = R\big(\mathcal{X} \cap \mathcal{G}^*\big) \oplus w. \qquad (14)$$

This has the following important consequence: We can eliminate spurious transitions by deciding whether the flowpipe intersects with $\mathcal{G}^*$. We call this *flowpipe separation*, and our approach is to reduce the problem to separating a number of convex sets, which is the topic of Sect. 4. The flowpipe separation will then be discussed in Sect. 5.

4. SEPARATING CONVEX SETS USING SUPPORT FUNCTIONS

A classic way to show that two convex sets do not overlap is to find a hyperplane that separates them (the sets lie on opposites sides of the plane). Efficient algorithms for finding a separating hyperplane are known, e.g., closest points algorithms like the *Gilbert-Johnson-Keerthi (GJK) algorithm* or the *Chung-Wang algorithm*, see [12]. We refer to these as *convex separation algorithms.* In this section, we propose convex separation algorithms that differ in two aspects:

- We consider the case where only the value of the support function can be computed, while classical methods are based on computing points in the set (support vectors).
- We take into account that the support function is computed with finite accuracy, i.e., up to an interval that contains the exact value.

The following well-known lemma expresses separation with support functions.

Lemma 4.1 (Separation of convex sets). *Given two compact convex sets $\mathcal{R}, \mathcal{S}$, let $\mathcal{Q} = \mathcal{R} \oplus (-\mathcal{S})$, i.e.,$\rho_\mathcal{Q}(d) = \rho_\mathcal{R}(d) + \rho_\mathcal{S}(-d)$. $\mathcal{R}$ and $\mathcal{S}$ are separated if and only if $0 \notin \mathcal{Q}$, or, equivalently, there is a $d^* \in \mathbb{R}^n$ with*

$$\rho_\mathcal{Q}(d^*) < 0. \qquad (15)$$

If d^* exists, any hyperplane $\mathcal{H} = \{x \mid {d^*}^\mathsf{T} x = b\}$ with b in the open interval $(\rho_\mathcal{R}(d^*), -\rho_\mathcal{S}(-d^*))$ separates $\mathcal{R}$ and $\mathcal{S}$.

In the following, we present separation algorithms adapted to approximately computing support functions.

4.1 Separation using Directed Approximation

We now propose a procedure for deciding the separation problem, based on iteratively constructing inner- and outer approximations of $\mathcal{Q}$. It is based on a polyhedral approximation algorithm called *Mutually Converging Polytopes* (MCP) by Kamenev [9], which approximates a convex set with the asymptotically optimal number of evaluations of the support function.

Given $\mathcal{Q}$ and a given number of iterations k_max, the MCP algorithm constructs an outer approximation Q_k with at most k facets and an inner approximation C_k with at most k vertices as follows:

1. Start with $n + 1$ affinely independent directions d_i. In each direction d_i, compute the support vector c_i of $\mathcal{Q}$. Let $k := n + 1$.
2. Compute the outer approx. $Q_k := \bigcap_{i=1}^k \{d_i^\mathsf{T} x \le d_i^\mathsf{T} c_i\}$.
3. Compute the inner approx. $C_k := \text{CH}(c_1, \ldots, c_k)$ in constraint representation, and let L be its set of constraints.
4. For each constraint $a_i^\mathsf{T} x \le b_i$ in L, compute the directional distance δ_i between the inner and the outer approximation, $\delta_i := (\rho_{\mathcal{Q}_k}(a_i) - b_i)/||a_i||$.
Let $d_{k+1} := a_{i_\text{max}}$ with $i_\text{max} = \text{argmax}_i\, \delta_i$.

5. Compute the support vector in the new direction d_{k+1}.
6. If $k = k_{\max}$, stop. Otherwise, let $k := k + 1$ and go to step 2.

The MCP algorithm has optimal convergence rate, see [9] for details. The Haussdorff distance between the outer and inner approximation is bounded by the value of $\delta_{i_{\max}}$, and converges to 0; in this sense, the algorithm is *complete*.

The main steps of the MCP algorithm are inherited by our algorithm, but it differs in three important ways:

- Instead of support vectors, we use an inner *estimation*, i.e., points which might not actually be in $\mathcal{Q}$. This makes the algorithm applicable to using only support function values and to approximate computations.
- The inner *estimation* is used for choosing the next direction, while the inner *approximation* (points which are known to be in $\mathcal{Q}$), is used only as a termination criterion in case of overlap.
- We refine only in directions that are still necessary to decide whether $\mathcal{Q}$ contains 0.

We use the following notation: Throughout, we use the index k to indicate the iteration. Let d_k be the direction in which the approximation is refined in the k-th iteration. Let r_k^+ be a bound on the support of $\mathcal{Q}$ in direction d_i, and with accuracy ε_k, $r_k^+ = \mathsf{support}(\mathcal{Q}, d_k, \varepsilon_k)$. Let Q_k be the outer approximation, i.e.,

$$Q_k = \lceil \mathcal{Q} \rceil_{D_k} = \bigcap\nolimits_{i=1}^{k} \{d_i^{\mathsf{T}} x \leq r_i^+\}.$$

Let $S_{k,i}$ be the facet slab of Q_k in direction d_i,

$$S_{k,i} = Q_k \cap \{d_i^T x \geq r_i^+ - \varepsilon_i\},$$

and let $c_{k,i}$ be a point in $S_{k,i}$ lying on a facet of Q_k, i.e.,

$$c_{k,i} \in S_{k,i} \cap \{d_i^T x \geq \rho_{Q_k}(d_i)\}.$$

Note that $c_{k,i}$ can be any point in $S_{k,i}$, e.g., the relative Chebyshev center. We choose them on the border of Q_k because this allows for an efficient, incremental, construction of their convex hull. Let $C_k = \mathrm{CH}(c_{k,1}, ..., c_{k,k})$ be the convex hull of the centers represented in constraint form. Let e_i be the n-dimensional vector with its i-th entry being 1 and all other entries being zero. Let $\varepsilon \geq 0$ be the accuracy used when evaluating the support function evaluation, and let $\varepsilon_{\min} \geq 0$ be a minimum accuracy that serves as termination criterion in case separation can not be decided.

Our *Directed Approximation* algorithm takes as inputs $\mathcal{Q} = \mathcal{R} \oplus (-\mathcal{S})$, an initial accuracy ε_0, a termination threshold accuracy $\varepsilon_{\min}$, and an eagerness parameter $\alpha > 1$ that represents the trade-off between sampling more directions and using a higher accuracy. The algorithm proceeds as follows:

1. Initialization: Choose as initial directions the normal vectors of a regular simplex: Let $d_i := e_i$ for $i = 1, ..., n$, and $d_{n+1} := -\sum_{i=1}^{n} e_i$. Let $k := n + 1$. Compute $r_i^+ = \mathsf{support}(\mathcal{Q}, d_i, \varepsilon_i)$ for $i = 1, \ldots, k$, with $\varepsilon_i := \varepsilon_0$.
2. Construct the outer approximation Q_k, its facet slabs $S_{k,1}, \ldots, S_{k,k}$, and points on the facets $c_{k,1}, ..., c_{k,k}$.
3. Compute the convex hull C_k in constraint representation. Decide, which constraints of C_k are relevant by measuring the directional distance between the inner approximation and zero. The constraints are contracted to obtain an inner approximation of $\mathcal{Q}$.
    (a) For each constraint $a_i^{\mathsf{T}} x \leq b_i$ of C_k do
        i. $J_i := \{j \mid a_i^{\mathsf{T}} c_j = b_i\}$. (indices of adjacent c_i)
        ii. $b_i^- := \min_{j \in J_i} -\rho_{S_{k,i}}(-a_i)$.
    (b) Let $L = \{a_i^{\mathsf{T}} x \leq b_i^- \mid b_i^- < 0\}$. (constraints already satisfied by $x = 0$ need not be refined)
    (c) If $L = \{\}$, stop with result "overlap"
4. Decide in which direction to refine, based on the distance δ between the relevant constraints and the outer approximation.
    (a) For each constraint $a_i^{\mathsf{T}} x \leq b_i$ in L, let $\delta_i := (\rho_{Q_k}(a_i) - b_i)/||a_i||$.
    (b) Let $d_{k+1} := a_{i_{\max}}$ with $i_{\max} = \mathrm{argmax}_i\, \delta_i$.
    (c) If $\delta_{i_{\max}} \leq \alpha\varepsilon_k$, let $\varepsilon_{k+1} := \varepsilon_k/10$, else $\varepsilon_{k+1} := \varepsilon_k$.
    (d) If $\delta_{i_{\max}} \leq \varepsilon_{\min}$, stop with result "unknown".
5. Compute an upper bound on the support function in the new direction and with maximum error ε.
    (a) $r_{k+1}^+ := \mathsf{support}(\mathcal{Q}, d_{k+1}, \varepsilon)$.
    (b) If $r^+ < 0$, stop with result "separation".
6. Let $k := k + 1$ and go to step 2.

The eagerness parameter α is motivated as follows: Even assuming that C_k converges to within distance ε_k of $\mathcal{Q}$ (we have no guarantee), we have that $\delta_{i_{\max}} \to \varepsilon_k$, which may lead to infinitely many iterations without ever satisfying $\delta_{i_{\max}} \leq \varepsilon_k$. Thus we must decrease ε_k at some point while $\delta_{i_{\max}} > \varepsilon_k$ still holds, which is guaranteed by choosing $\alpha > 1$. Larger values of α lead to a faster decrease of ε_k.

LEMMA 4.2. *There result of the Directed Approximation algorithm is sound if it returns "separation" or "overlap". If it returns "unknown", the distance between $\mathcal{R}$ and $\mathcal{S}$ is bounded above by*

$$\delta = \min_x \|x\|^2 \ \text{s.t.} \ \bigcap_{i=1}^{k} \{a_i^{\mathsf{T}} x \leq b_i^-\}.$$

PROOF. The soundness of the result "separation" follows directly from (15). The soundness of the result "overlap" follows from Prop. 2.2, which states that $C^- = \bigcap_{i=1}^{k} \{a_i^{\mathsf{T}} x \leq b_i^-\} \subseteq \mathcal{Q}$. L contains all $a_i^{\mathsf{T}} x \leq b_i^-$ with b_i^-. Since "overlap" results only when $L = \{\}$, we have that all $b_i^- > 0$, and therefore $0 \in C^- \subseteq \mathcal{Q}$. With Prop. 4.1, this implies overlap.

With $C^- \subseteq \mathcal{Q}$ it follows that $\delta \geq \min_q \|q\|^2$ s.t. $q \in \mathcal{Q}$. Let q^* be such a minimizer. Since $\mathcal{Q} = \mathcal{R} \oplus (-\mathcal{S})$, this means that there exists $r \in \mathcal{R}$ and $s \in \mathcal{S}$ such that $q^* = r - s$, and therefore $\|r - s\|^2 \leq \delta$. □

A demonstration of the directed approximation algorithm can be seen in Fig. 2. On the left hand side, $\mathcal{R}$ and $\mathcal{S}$ are shown. On the right hand side, the according $\mathcal{Q}_k$ and all $\mathcal{C}_k$ are shown. Fig. 3 shows the set of $\mathcal{C}_k$ in the iteration before finding the separation. We observe that most of the points are concentrated around the origin.

4.2 Adapted GJK Algorithm

Given compact convex sets $\mathcal{R}, \mathcal{S}$, a closest point algorithm computes the (not necessarily unique) pair of points $r^* \in \mathcal{R}$

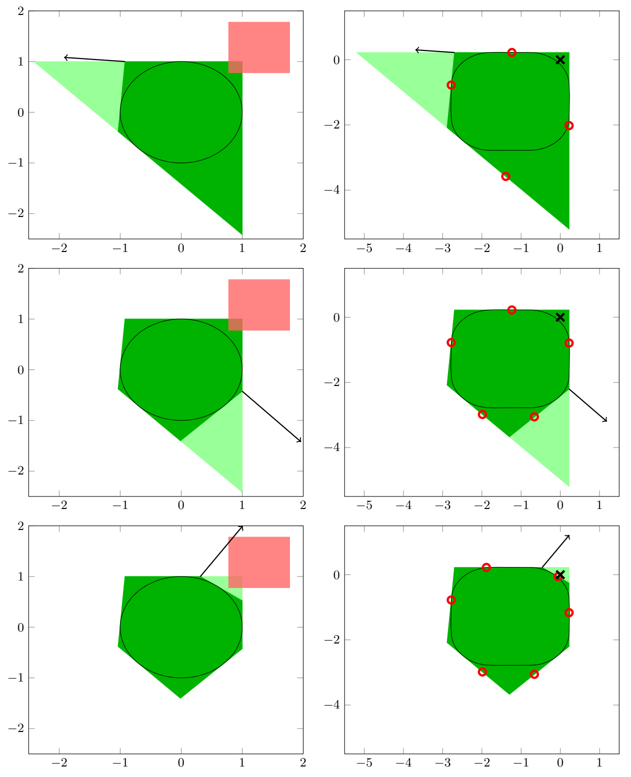


Figure 2: Demonstration of the directed approximation algorithm (top to bottom). The left column shows the set $\mathcal{R}$ (black outline), its overapproximation (dark green), and the guard set $\mathcal{S}$ (red box). The right column shows $\mathcal{Q} = \mathcal{R} \oplus (-\mathcal{S})$ (black outline) the outer approximation Q_k (dark green) and the vertices of the inner approximation C_k (red circles). The last iteration shows separation since the origin (black x) lies outside of Q_k

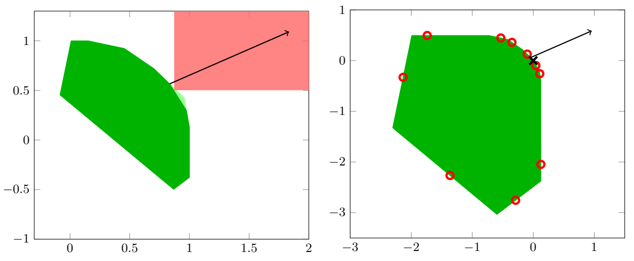


Figure 3: Example that shows the directed fashion of the algorithm: facets in the lower left hand corner of $\mathcal{R}$ are no longer refined, since it was shown that refining them will not improve the separation result.

and $s^* \in \mathcal{S}$ that are closest to each other. Finding such r^*, s^* can be reduced to finding the (unique) $q^* \in \mathcal{Q}$ in closest to 0. If $q^* = 0$, then $\mathcal{R}$ and $\mathcal{S}$ overlap. Otherwise, $d = q^*$ is the normal vector of a separating hyperplane as in Lemma 4.1.

The *Gilbert-Johnson-Keerthi (GJK) algorithm* finds such a q^* iteratively by computing maximizers. It takes advantage of the following property: Any $q \in \mathcal{Q}$ is closest to 0 if and only if q is the minimizer of $\mathcal{Q}$ in direction $d = q$, and this point is unique. Note that a minimizer of $\mathcal{Q}$ in direction d is a maximizer (support vector) of $\mathcal{Q}$ in direction $-d$. A rudimentary form of the algorithm goes as follows:

1. Start from an arbitrary direction d_0. Let $k = 0$.
2. Compute a point q_k that maximizes $d_k^\mathsf{T} q$ for $q \in \mathcal{Q}$.
3. Let q_k^* be the point in $\mathrm{CH}\{q_0, \ldots, q_k\}$ closest to 0, and let $d_{k+1} = -q_k^*$.
4. If $d_k^\mathsf{T} d_{k+1} = \|d_k\|\|d_{k+1}\|$, then stop. The point in $\mathcal{Q}$ closest to 0 is q_k^*.
5. Let $k \leftarrow k + 1$ and go to step 2.

The GJK algorithm is guaranteed to converge towards the closest point, and terminate if $\mathcal{Q}$ is a polytope. Note that if $0 \in \mathrm{CH}\{q_0, \ldots, q_k\}$, then $\mathcal{R}$ and $\mathcal{S}$ overlap, and the algorithm terminates with $q_k^* = 0$. The termination criterion in step 4 is usually relaxed to

$$|d_k^\mathsf{T} d_{k+1} - \|d_k\|\|d_{k+1}\|| \leq \varepsilon$$

for some given tolerance level $\varepsilon \geq 0$. If one is only interested in showing separation, the criterion (15) can be used to terminate early. Several efficiency improvements are known, but are omitted here for lack of space.

The GJK algorithm is not directly applicable in our setting, because we can only compute approximate support functions, not the corresponding support vectors. We now present a variation of the GJK algorithm that is solely based on approximate support functions.

Because we can not compute maximizers of $\mathcal{Q}$, we use centers of facet slabs instead. Since these points may not actually be in $\mathcal{Q}$, we must find new directions even if $0 \in \mathrm{CH}\{q_0, \ldots, q_k\}$. In this case, we choose the closest point on the *border* of $\mathrm{CH}\{q_0, \ldots, q_k\}$, which tends to "push" facets outwards in a way similar to the Directed Approximation algorithm. Since we need bounded facet slabs, we start with a bounded initial approximation. Given the set $\mathcal{Q}$ and a termination threshold accuracy $\mu_{\min} \geq 0$, our modified GJK algorithm proceeds as follows:

1. Construct an initial, bounded outer approximation to get facet slabs.
   (a) Let $D_{init} = \{d_0, \ldots, d_{m-1}\}$ be a set of directions that span $\mathbb{R}^n$, e.g., the normals of a regular simplex or a bounding box.
   (b) Start from an arbitrary direction d_m. Let $k = m$.
2. *Estimate* a point q_k that maximizes $d_k^\mathsf{T} q$ for $q \in \mathcal{Q}$.
   - Compute $r_k^+ = \mathsf{support}(\mathcal{Q}, d_k, \varepsilon)$. If $r_k^+ < 0$, stop with result "separation".
   - Choose $q_k \in \lfloor \mathcal{Q} \rceil_k$, e.g., the relative Chebyshev center.
3. Let q_k^* be the point *on the border* of $\mathrm{CH}\{q_m, \ldots, q_k\}$ closest to 0. If $0 \notin \mathrm{CH}\{q_m, \ldots, q_k\}$, let $d_{k+1} = -q_k^*$ (like GJK). Otherwise, let $d_{k+1} = q_k^*$.
4. If $d_{k+1} \in D$, abort since an infinite cycle may take place. Otherwise, add d_{k+1} to D.
   If $|d_k^\mathsf{T} d_{k+1} - \|d_k\|\|d_{k+1}\|| \leq \mu_{\min}$, stop with result "unknown".
5. Let $k \leftarrow k + 1$ and go to step 3.

This modified GJK algorithm may not terminate, or even converge to the point closest to 0. It is presented here because it can detect separation often much faster than Directed Approximation. This will be examined closer in the experimental section.

5. TIMED FLOWPIPE SEPARATION

The timed flowpipe separation problem is to identify the time points where the flowpipe is separated from a given (guard) set $\mathcal{S}$. We limit our discussion to a bounded guard set $\mathcal{S}$ and a finite time horizon T. If $\mathcal{S}$ is unbounded, one can render it bounded by computing a coarse flowpipe approximation that is bounded due to finite T, and intersecting $\mathcal{S}$ with this coarse approximation.

Definition 5.1 (Timed flowpipe separation). *Given compact convex sets $\mathcal{X}_0, \mathcal{S} \subset \mathbb{R}^n$ and a time interval $[t_1, t_2]$, a* separating time domain *$\mathcal{T}$ is a subset of $[t_1, t_2]$ such that for all $t \in \mathcal{T}$, $\mathcal{X}_t \cap \mathcal{S} = \emptyset$.*

Knowing the time intervals in which the system enters and leaves the guard can be used to improve the flowpipe approximation. Similarly, timed flowpipe separation can identify at what time t' all trajectories have left the invariant $\mathcal{I}$. Then t' can be taken as time horizon for a more precise flowpipe approximation. The smaller $\mathcal{T}$, the more precise (and cheaper) the flowpipe approximations can be.

In this section, we present algorithms to decide flowpipe separation with as little computational effort as possible.

5.1 Flowpipe Separation using Convexification

Flowpipe separation using convexification is a straightforward application of the convex separation algorithms to a flowpipe approximation consisting of a finite number of convex sets. For each set in the approximation, a convex separation algorithm is executed. If it shows separation on all sets in the sequence, the flowpipe is separated. However, it must be decided when a convex set is accurate enough, or whether it requires being split in several parts.

We now present a separation procedure for a given initial set $\mathcal{X}_0$, a guard set $\mathcal{S}$ and a time interval $[t_1, t_2]$. It uses the convexified flowpipe approximation from Sect. 3.2 and a convex separation algorithm from Sect. 4.1, and returns a set of convex sets that could not be separated from $\mathcal{S}$.

1. Start with an initial accuracy ε_0 and an initial set of directions $D = \{d_1, \ldots, d_m\}$ that spans $\mathbb{R}^n$, which guarantees that the approximation is bounded.
2. Apply the flowpipe approximation from Sect. 3.2 to compute the flowpipe approximation consisting of convex sets $\Omega_0, \Omega_1, \ldots$, using directions D and accuracy ε_0.
3. For each Ω_j, run a convex separation algorithm to separate it from $\mathcal{S}$, where each call to $\mathsf{support}(\Omega_i, d, \varepsilon)$ is implemented as follows:
   (a) Compute upper and lower bounds on the support function of $\mathcal{X}_t$: $(s^+_{d,\varepsilon}(t), s^-_{d,\varepsilon}(t)) = \text{sReach}(\mathcal{X}_0, A, \mathcal{U}, [t_j, t_{j+1}], d, \varepsilon)$.
   (b) Let $\hat{s}(t)$ be the concave hull of $s^+_{d,\varepsilon}(t)$ over the time interval $[t_j, t_{j+1}]$. This corresponds to convexifying the set over this time interval.
   (c) Let $s^+ = \max_{t \in [t_j, t_{j+1}]} \hat{s}(t)$, let $\varepsilon_{\text{result}} = \max_{t \in [t_j, t_{j+1}]} \hat{s}(t) - s^-_{d,\varepsilon}(t)$.
   (d) If $\varepsilon_{\text{result}} \leq \varepsilon$, use s^+ as support value for the support function of Ω_j.
   (e) Otherwise, a single set does not suffice to represent the flowpipe with sufficient accuracy. Divide $[t_j, t_{j+1}]$ into subintervals such that on each interval, the concave hull of $s^+_{d,\varepsilon}(t)$ satisfies the conditions of step 3c and 3d. Replace Ω_j by restrictions of Ω_j to those subintervals. For each, apply the convex separation algorithm again.
4. Return the Ω_j, for which separation could not be shown.

5.2 Flowpipe Separation Point-Wise over Time

A convex separation algorithm can solve the flowpipe separation problem for any given point in time t^*, since we know that $\mathcal{X}_{t^*}$ is convex. However, we need to extend separation to intervals of time. With Lemma 4.1, the following criterion is straightforward.

Lemma 5.2. *The flowpipe $\mathcal{X}_{t_1,t_2}$ is separated from a convex set $\mathcal{S}$ if and only if for all $t \in [t_1, t_2]$ there exists a direction $d_t \in \mathbb{R}^n$ such that*

$$\rho_{\mathcal{X}_t}(d_t) + \rho_{\mathcal{S}}(-d_t) < 0. \tag{16}$$

The question is therefore how to find a suitable direction d_t for each point in time. We present two ways for applying separation over an interval of time: first, keeping the direction d fixed over time, and second, letting d_t evolve over time according to the dynamics of the system.

5.2.1 Separating with Fixed Direction

We use the bounds on the support function of $\mathcal{X}_t$ as described in Sect. 3.2. Given a time interval $[t_1, t_2]$, a direction d, and a precision $\varepsilon > 0$, we construct a piecewise linear function $s^+_{d,\varepsilon}(t) = \text{sReach}(\mathcal{X}_0, A, \mathcal{U}, [t_1, t_2], d, \varepsilon)$, so that for all $t \in [t_1, t_2]$, $\rho_{\mathcal{X}_t}(d) \leq s^+_{d,\varepsilon}(t)$. Applying (16), the flowpipe is separated from $\mathcal{S}$ for any $t \in [t_1, t_2]$ for which

$$s^+_{d,\varepsilon}(t) + \rho_{\mathcal{S}}(-d) < 0. \tag{17}$$

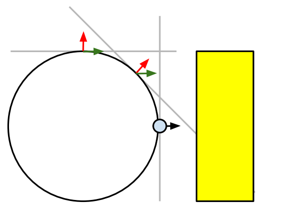
Note that the direction d is fixed and does not change with time.

The following example shall illustrate that there are cases where *only* a single, fixed, direction (or an arbitrarily small neighborhood around it) can show separation.

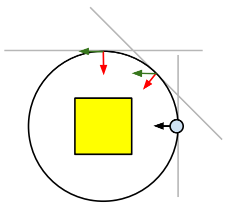
Example 5.3. *Consider the example shown in Fig. 4(a). The initial set $\mathcal{X}_0$ consists of a single point, and the flowpipe consists of the point moving around the origin in a circle. The direction $d = (1, 0)$ shows separation even over an unbounded time horizon, as indicated by the green arrows. By making $\mathcal{S}$ large enough in the vertical direction, we can reduce the set of separating directions to an arbitrarily small neighborhood of d. A guard may not be separable by a single fixed direction over the entire time horizon, as shown in Fig. 4(b).*

5.2.2 Separating with Dynamic Direction

Instead of keeping the direction fixed over time, we can let it evolve according to the dynamics of the system. Given a direction d_0, let $d_t = e^{-A^\mathsf{T} t} d_0$. The dynamic direction d_t evolves with the system in the following sense. If d_0 is a normal vector of $\mathcal{X}_0$ at a point $x^*(0)$ (meaning it is tangent to $\mathcal{X}_0$ and touches at x^*_0), then d_t is a normal vector of $\mathcal{X}_t$ at a point $x^*(t)$. E.g., if $\mathcal{X}_0$ is a polytope and d_0 is a facet normal of $\mathcal{X}_0$, then d_t is a facet normal of $\mathcal{X}_t$. The following example shall illustrate where a single dynamic direction can show separation over the entire time horizon.



(a) Keeping the direction that separates the initial set shows separation over the entire time horizon



(b) Dynamically adapting the direction that separates the initial set shows separation over the entire time horizon

Figure 4: In both examples, separation can be shown for the entire flowpipe after finding a separating direction for the initial set

EXAMPLE 5.4. *Consider the example shown in Fig. 4(a). Starting from a direction that separates the initial set, the dynamic direction (red arrows) shows separation for only a small amount of time, which is even smaller if the guard set is larger in the vertical direction. For the guard set shown in Fig. 4(b), the dynamic direction separates over the entire time horizon, while no fixed direction can do so.*

We can reduce the separation along a dynamic direction to the separation of a fixed direction, which allows us to apply the techniques from the previous section.

LEMMA 5.5. *Let $\hat{\mathcal{S}}_t = \text{Reach}_t(-\mathcal{S}, -A, \mathcal{U})$. $\mathcal{X}_t$ is separated from $\mathcal{S}$ in direction $d_t = e^{-A^\intercal t} d_0$ if and only if $\hat{\mathcal{S}}_t$ is separated from $\mathcal{X}_0$ in direction d_0. Indeed,*

$$\rho_{\mathcal{X}_t}(d_t) - \rho_{-\mathcal{S}}(d_t) = \rho_{\mathcal{X}_0}(d_0) + \rho_{\hat{\mathcal{S}}_t}(d_0).$$

PROOF. The proof is a simple calculation on support functions, using the solution (10) for the reachable set at time t. Note that through variable substitution one can show that

$$\int_0^t e^{As}\mathcal{U}ds = \int_0^t e^{A(t-s)}\mathcal{U}ds = e^{At}\int_0^t e^{-As}\mathcal{U}ds.$$

With $d_t = e^{-A^\intercal t} d_0$ and applying $\rho_R(M^\intercal d) = \rho_{MR}(d)$,

$$\begin{aligned}\rho_{\mathcal{X}_t}(d_t) + \rho_{-\mathcal{S}}(d_t) &= \rho_{\mathcal{X}_t}(e^{-A^\intercal t} d_0) + \rho_{-\mathcal{S}}(e^{-A^\intercal t} d_0)\\ &= \rho_{e^{-At}\mathcal{X}_t}(d_0) + \rho_{-e^{-At}\mathcal{S}}(d_0)\\ = \rho_{e^{-At}e^{At}\mathcal{X}_0}(d_0) &+ \rho_{e^{-At}e^{At}\int_0^t e^{-As}\mathcal{U}ds}(d_0) + \rho_{-e^{-At}\mathcal{S}}(d_0)\\ = \rho_{\mathcal{X}_0}(d_0) + \rho_{-e^{-At}\mathcal{S}\oplus\int_0^t e^{-As}\mathcal{U}ds}&(d_0) = \rho_{\mathcal{X}_0}(d_0) + \rho_{\hat{\mathcal{S}}_t}(d_0).\end{aligned}$$

□

The above lemma allows us to compute the separating time intervals for a dynamic direction as follows. Let $p^+_{d,\varepsilon}(t) = \text{sReach}(-\mathcal{S}, -A, \mathcal{U}, [t_1, t_2], d, \varepsilon)$, so that for all $t \in [t_1, t_2]$, $\rho_{\hat{\mathcal{S}}_t}(d) \leq p^+_{d,\varepsilon}(t)$. Applying Lemma 5.5 and (16), the flowpipe is separated from $\mathcal{S}$ for any $t \in [t_1, t_2]$ for which

$$\rho_{\mathcal{X}_0}(d) + p^+_{d,\varepsilon}(t) < 0. \qquad (18)$$

5.2.3 Pointwise Separation Algorithm

We now describe an algorithm that uses a convex separation algorithm pointwise in time to separate the flowpipe from a guard set $\mathcal{S}$ over a time interval $[t_1, t_2]$.

The algorithm takes as input the system description, the initial set $\mathcal{X}_0$, the guard set $\mathcal{S}$, a time interval $[t_1, t_2]$. It returns a set of time intervals for which separation could not be shown.

1. Picking some $t^* \in [t_1, t_2]$, e.g., the midpoint, we use a convex separation algorithm on $\mathcal{X}_{t^*}$ to detect or refute separation at time t^*. If separation cannot be shown, stop. Otherwise, we obtain a separating direction d and a bound ε on the required accuracy.
2. Compute $s^+_{d,\varepsilon}(t) = \text{sReach}(\mathcal{X}_0, A, \mathcal{U}, [t_1, t_2], d, \varepsilon)$ and $p^+_{d,\varepsilon}(t) = \text{sReach}(-\mathcal{S}, -A, \mathcal{U}, [t_1, t_2], d, \varepsilon)$.
3. Remove from $[t_1, t_2]$ the t where $s^+_{d,\varepsilon}(t) + \rho_{\mathcal{S}}(-d) < 0$ or $\rho_{\mathcal{X}_0}(d) + p^+_{d,\varepsilon}(t) < 0$ (separation holds).
4. For each of the remaining sub-intervals, apply the pointwise separation algorithm recursively and return the obtained intervals.

The algorithm has the weakness that it may stop prematurely if the separation time t^* is poorly chosen. We propose two improvements: First, the algorithm may be repeated on the subintervals $[t_1, t^*]$ and $[t^*, t_2]$, until their size falls below a given threshold. Second, the algorithm may be applied a second (and third) time, choosing t^* to be the start (and end) times of the intervals instead.

Indeed, separating on start and end times may reduce the size of the flowpipe segments, for which discrete successor states are computed, and thus improve the approximation accuracy of the reachability algorithm even in cases where separation could not be shown.

6. EXPERIMENTAL RESULTS

In this section, we evaluate the presented algorithms on two classes of benchmarks. We have implemented the algorithms in the SpaceEx hybrid model checker. We conducted the experiments on a machine with an Intel i7 3.4 GHz processor and 16 GB of RAM. We illustrate the results for the convex separation algorithms from Section 4 on the *sphere* benchmark. Here, we consider an overapproximation of an n-dimensional sphere by a polytope with m constraints. The guard is a single point. Recall that with Lemma 4.1, any convex separation problem can be reduced to separating one convex set from a single point (the origin). Our benchmark is thus equivalent to separating two sphere-approximations (each with half the radius), which we consider to be a challenging instance.

We consider a number of benchmark instances by varying the dimension of the sphere n, the number of constraints m as well as the distance to the guard. By varying those parameters we can flexibly adjust the benchmark instance complexity. The guard is defined based on the distance and the normal of the guard hyper-plane. For every tuple $(n, m, \text{distance})$ we pick 10 random vectors to be used as guard normals. We analyze every instance using the adapted GJK algorithm and the directed approximation algorithm. Note that we accumulate the results over those randomly selected vectors. In particular, we report the relation of the number of the benchmark instances where the convex separation algorithm has found the right answer before the time-out (success rate), the minimum, maximum and average numbers of direction refinements and run-time, respectively (see Table 1). We terminate the analysis when either the timeout of 500 s or the maximum number of 500 refinement iterations has been reached.

We observe that the time needed to show separation generally increases with the sphere dimensionality and the number of constraints. Furthermore, the number of direction refinements to show separation increases as well. The distance to the guard plays an important role: The smaller the distance, the more complex it becomes to find a separating plane. The success rate goes from 5% for instance 24 with the distance equal to 0.1 up to 100% for distance 1.

The GJK algorithm proves separation in most cases in a short time. The DA algorithm performed very well for the smaller dimensions and number of constraints, still scales worse than the GJK algorithm. We note that the DA algorithm provides more guarantees compared to the GJK algorithm. Furthermore, the DA algorithm can provide sound results for the cases with overlap.

We examine the flowpipe separation algorithms from Section 5 based on the circle benchmark. In this setting, we iteratively call the convex separation algorithm for every time interval where the system is expected to enter and leave the guard. In the circle benchmark, we model an object which moves on a circle orbit in 2D space, i.e, its dynamics are provided by the following differential equations: $\dot{x} = -y \wedge \dot{y} = x$. The system behavior is illustrated in Figure 4. We take a segment between two points on the orbit as an initial region. We consider two positions of a rectangular guard: the guard is inside the circular orbit (GI; see Figure 4(b)) and the guard is outside (GO; see Figure 4(a)). The results for the circle benchmark are presented in Table 2.

We observe that with the GO-instances, we need to consider much less time intervals. Furthermore, the number of directions to prove the separation drastically differs. For example, the directed approximation algorithm with flowpipe separation using convexification needs 5 direction refinements if the guard is outside. However, the number of refinements increases to 108 if the guard is located inside. Note that the algorithm also cannot prove the separation in this case. We can explain this behavior as follows. For the GO-instances, there exists at least one separating plane for *all* the time intervals. At the same time, if the guard is located inside, we observe that *no* plane exists which separates the guard from the flow-pipe for all time intervals. Therefore, the algorithm needs to search for an *individual* separating plane for every time interval for the GI-instances. Therefore, we expect the flowpipe separation using convexification to be useful for GO-instances. This hypothesis is confirmed by our experiments where all the convex separating algorithms require only one interval refinement. However, due to the same reasons, the flowpipe separation performs badly on the GI-instances. Moreover, the convex separating algorithms cannot even find the separating plane because the convexification leads to rather over-approximative results.

ID	Algorithm	# IR	# d	Runtime	Res.
1	CH, GJK	1	7	0.038s	SEP
2	CH, DA	1	5	0.046s	SEP
3	PW, GJK	1	6	0.042s	SEP
4	PW, DA	1	6	0.060s	SEP
5	CH, GJK	9	1767	37.903s	INT
6	CH, DA	14	108	0.749s	INT
7	PW, GJK	19	258	0.960s	SEP
8	PW, DA	18	128	0.791s	SEP

Table 2: Experimental results of the circle benchmark with static interval refinement. Abbreviations: ID: benchmark instance number, Algorithm: the convex separation algorithm used, # IR: number of interval refinements, # d: number of considered directions over all the interval refinements, Runtime: algorithm runtime, Res.: result of the algorithm. Note that the separating plane exists in all the instances. The GO-instances are in the upper block of the table. The GI-instances are in the lower block. CH stands for the flowpipe separation using convexification. PW stands for the pointwise flowpipe separation. GJK stands for the adapted GJK algorithm. DA denotes the directed approximation algorithm.

7. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[2] R. Alur, T. Dang, and F. Ivancic. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006.

[3] E. Asarin, T. Dang, O. Maler, and R. Testylier. Using redundant constraints for refinement. In *Automated Technology for Verification and Analysis*, pages 37–51. Springer, 2010.

[4] P. S. Duggirala and A. Tiwari. Safety verification for linear systems. In *EMSOFT'13*, pages 1–10. IEEE, 2013.

[5] G. Frehse, R. Kateja, and C. Le Guernic. Flowpipe approximation and clustering in space-time. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 203–212. ACM, 2013.

[6] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011.

[7] G. Frehse and R. Ray. Flowpipe-guard intersection for reachability computations with support functions. In *IFAC ADHS*, pages 94–101, 2012.

[8] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *LNCS*, pages 540–554. Springer, 2009.

[9] A. V. Lotov, V. A. Bushenkov, and G. K. Kamenev. *Interactive Decision Maps*, volume 89 of *Applied Optimization*. Kluwer, 2004.

[10] I. M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. In *HSCC'07*, pages 428–443, 2007.

[11] S. Sankaranarayanan, T. Dang, and F. Ivančić. Symbolic model checking of hybrid systems using template polyhedra. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–202. Springer, 2008.

[12] G. van den Bergen. *Collision detection in interactive 3D computer animation.* PhD thesis, Eindhoven University of Technology, 1999.

ID	Algorithm	n	m	Distance	✓ %	# Direction Ref.			Runtime		
						min.	max.	avg.	min.	max.	avg.
1	GJK	2	16	0.01	100%	2	52	13.550	0s	0.077s	0.015s
2	GJK	3	36	0.01	95%	7	54	30.789	0.008s	0.125s	0.061s
3	GJK	4	64	0.01	85%	20	110	67.588	0.056s	0.895s	0.446s
4	GJK	5	100	0.01	55%	105	162	133.909	1.402s	6.689s	3.027s
5	GJK	2	16	0.1	95%	1	19	7.157	0s	0.021s	0.005s
6	GJK	3	36	0.1	100%	2	34	12.000	0s	0.067s	0.017s
7	GJK	4	64	0.1	95%	2	56	18.263	0.001s	0.241s	0.057s
8	GJK	5	100	0.1	65%	9	40	18.833	0.027s	0.288s	0.104s
9	GJK	2	16	0.5	100%	1	3	1.900	0s	0.002s	0s
10	GJK	3	36	0.5	100%	1	11	3.800	0s	0.015s	0.003s
11	GJK	4	64	0.5	100%	1	21	6.300	0s	0.057s	0.013s
12	GJK	5	100	0.5	95%	2	27	7.000	0.001s	1.300s	0.089s
13	GJK	2	16	1	100%	1	2	1.650	0s	0.001s	0s
14	GJK	3	36	1	100%	1	3	2.150	0s	0.002s	0s
15	GJK	4	64	1	100%	1	8	2.850	0s	0.015s	0.002s
16	GJK	5	100	1	100%	2	8	3.150	0.001s	0.026s	0.004s
17	DA	2	16	0.01	100%	2	10	6.500	0.003s	0.047s	0.023s
18	DA	3	36	0.01	95%	30	116	46.368	0.707s	33.642s	3.553s
19	DA	4	64	0.01	40%	150	210	180.875	142.302s	470.624s	309.943s
20	DA	5	100	0.01	0%	500	500	—	—	0s	—s
21	DA	2	16	0.1	100%	2	7	3.750	0s	0.027s	0.008s
22	DA	3	36	0.1	100%	7	40	18.800	0.030s	1.421s	0.306s
23	DA	4	64	0.1	95%	13	161	88.210	0.207s	188.231s	51.952s
24	DA	5	100	0.1	5%	116	116	116.000	188.580s	188.580s	188.580s
25	DA	2	16	0.5	100%	2	4	2.450	0s	0.010s	0.002s
26	DA	3	36	0.5	100%	2	13	5.500	0s	0.103s	0.028s
27	DA	4	64	0.5	100%	2	83	22.900	0.001s	23.274s	2.704s
28	DA	5	100	0.5	85%	2	113	28.000	0.001s	179.145s	20.968s
29	DA	2	16	1	100%	2	2	2.000	0s	0.002s	0s
30	DA	3	36	1	100%	2	10	3.150	0s	0.079s	0.009s
31	DA	4	64	1	100%	2	31	7.300	0s	1.396s	0.182s
32	DA	5	100	1	100%	2	40	9.050	0s	7.546s	0.671s

Table 1: Results for the sphere benchmark. Abbreviations: ID: benchmark instance ID, Algorithm: convex separation algorithm, n: dimension of the sphere, m: number of facets in the over-approximation of the sphere, Distance: distance between the guard and the polytope which over-approximates the sphere, ✓ %: the relation of the number of the benchmark instances where a convex separation algorithm has found the right answer (until OOT = 500 s.) to the total number of benchmarks in this class (success rate), # Direction Ref.: number of directions the separation algorithm has tried out (minimum, maximum and average values), Runtime: runtime in seconds (minimum, maximum and average values), DA denote the directed approximation algorithm, GJK stands for the adapted GJK algorithm.