Noname manuscript No.
(will be inserted by the editor)

Unbounded-Time Safety Verification of Guarded LTI Models with Inputs by Abstract Acceleration

Dario Cattaruzza · Alessandro Abate · Peter Schrammel · Daniel Kroening

the date of receipt and acceptance should be inserted later

Abstract Reachability analysis of dynamical models is a relevant problem that has seen much progress in the last decades, however with clear limitations pertaining the nature of the dynamics and the soundness of the results. This article focuses on sound safety verification of unbounded-time (infinite-horizon) Linear Time-Invariant (LTI) models with inputs using reachability analysis. We achieve this using counterexample-guided Abstract Acceleration: this approach over-approximates the reachability tube of the LTI model over an unbounded time horizon by using abstraction, possibly finding concrete counterexamples for refinement based on the given safety specification. The technique is applied to a number of LTI models and the results show robust performance when compared to state-of-the-art tools.

1 Introduction

Linear loops are an ubiquitous programming pattern [47]. Linear loops iterate over continuous variables (in the case of physical systems) or discrete variables (in the case of loops in digital programs), which are updated with a linear transformation. Linear loops may be guarded, i.e., halt if a given linear condition holds: at this point the system may either elicit a new behaviour, or simply terminate. Inputs from the environment can be modelled by means of non-deterministic choices within the loop. These features make linear loops expressive enough to capture the dynamics of many hybrid dynamical models [6, 47]. The usage of such models in safety-critical embedded systems makes linear loops a fundamental target for formal methods.

Many high-level requirements for embedded control systems can be modelled as safety properties, *i.e.* deciding reachability of certain *bad states*, for which the model exhibits unsafe behaviour. Bad states may, in linear loops, be encompassed by guard assertions, namely (linear) constraints over their continuous variables. Reachability in linear programs, however, is a formidable challenge for automatic analysers: despite the restriction to linear transformations (i.e., linear dynamics) and linear guards, it is in the general case undecidable. The problem has been related to the Skolem Problem, which includes a subset of proven decidable cases (eg. the orbit problem [46], and for low order models [52]). The problem is decidable when reduced to finite state spaces, but even then, algorithms are costly (in particular for the infinite time case).

The goal of this article is to push the frontiers of unbounded-time reachability analysis: we aim at devising a method that is able to reason soundly about unbounded trajectories by performing *abstract acceleration*. Abstract acceleration [36, 37, 44] approximates the effect of an arbitrary number of loop iterations (up to infinity) with a single, non-iterative transfer function that is applied to the entry state of the loop (i.e., to the set of initial

Address(es) of author(s) should be given

conditions of the linear dynamics). This article extends the work in [44] to models with non-deterministic inputs elaborating an early work in [55] and completing [14, 16].

The key original contributions of this article are as follows:

1. We present a new technique to include time-varying non-deterministic inputs in the abstract acceleration of general linear loops.
2. We extend abstract acceleration to the continuous time case.
3. We introduce a technique to help the analysis of support functions in complex spaces, in order to increase the precision of previous abstract acceleration methods.
4. We develop a counterexample-guided refinement for Abstract Acceleration for safety verification, maximising speed when high precision is not necessary, thus allowing for optimal analysis within a safe region.
5. We provide an implementation of the discussed procedures as a tool, called *Axelerator*, which is available at

   `http://www.cprover.org/LTI/`

   We test the novel procedures over a broad set of experiments: these benchmarks (including specifications of the initial states, input ranges and guard sets), are also available.

Finally, in Section 10 we provide a thorough review of and comparison with related work.

2 Preliminaries

2.1 Linear loops with inputs - syntax

A discrete time LTI model may be described as a simple linear loop. Simple linear loops are functions expressed in the form:

$$\textit{while}(\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}) \;\; \boldsymbol{x} := \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u},$$

where $\boldsymbol{x} \in \mathbb{R}^p$ is a valuation on the state variables, $\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}$ is a linear constraint on the states (with $\boldsymbol{G} \in \mathbb{R}^{r \times p}$ and $\boldsymbol{h} \in \mathbb{R}^r$), $\boldsymbol{u} \in \mathbb{R}^q$ is a non-deterministic input, and $\boldsymbol{A} \in \mathbb{R}^{p \times p}$ and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$ are linear transformations characterising the dynamics of the model. This syntax can be interpreted as the dynamics of a discrete-time LTI model with inputs, under the presence of a guard set which, for ease of notation, we denote as $G = \{\boldsymbol{x} : \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\}$. The purpose of the guard is to restrict the dynamics to a given set, either to ensure safety (think for example of speed limits) and/or to change the behaviour of the model under certain conditions (e.g. once we have reached a certain state we begin a new process).

In particular, the special instance where $G = \top$ (i.e., "while true") represents a time-unbounded loop with no guards, for which the discovery of a suitable invariant (when existing) is paramount.

2.2 Model semantics

The traces of the model starting from an initial set $X_0 \subseteq \mathbb{R}^p$, with inputs restricted to the set $U \subseteq \mathbb{R}^q$, are sequences $\boldsymbol{x}_0 \xrightarrow{\boldsymbol{u}_0} \boldsymbol{x}_1 \xrightarrow{\boldsymbol{u}_1} \boldsymbol{x}_2 \xrightarrow{\boldsymbol{u}_2} \ldots$, where $\boldsymbol{x}_0 \in X_0$ and $\forall k \geq 0, \boldsymbol{x}_{k+1} = \tau(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and $\boldsymbol{u}_k \in U$, satisfying

$$\tau(\boldsymbol{x}_k, \boldsymbol{u}_k) = \{\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \text{ given } \boldsymbol{G}\boldsymbol{x}_k \leq \boldsymbol{h}\}. \tag{1}$$

We extend the point-wise notation above to convex sets of states and inputs (X_k and U), and denote the set of states reached from set X_k by τ in one step as

$$\tau(X_k, U) = \{\tau(\boldsymbol{x}_k, \boldsymbol{u}_k) : \boldsymbol{x}_k \in X_k, \boldsymbol{u}_k \in U\}. \tag{2}$$

We furthermore denote the set of states reached from X_0 via τ in n steps (*n-reach set*, constrained by G), for $n \geq 0$:

$$\tau^0(X_0, U) = X_0, \quad \tau^n(X_0, U) = \tau(\tau^{n-1}(X_0, U) \cap G, U). \tag{3}$$

Since the sets X_0 and U are convex, the transformations $\boldsymbol{A}$ and $\boldsymbol{B}$ are linear, and vector sums preserve convexity, the sets $X_n = \tau^n(X_0, U)$ are also convex.

We define the *n-reach tube*

$$\hat{X}_n = \hat{\tau}^n(X_0, U) = \bigcup_{k \in [0,\dots,n]} \tau^k(X_0, U) \tag{4}$$

as the union of k-reach sets over n iterations. Moreover, $\hat{X} = \bigcup_{n \geq 0} \tau^n(X_0, U)$ extends the previous notion to an unbounded time horizon (transitive closure).

2.3 Spectral eigendecomposition

Eigendecomposition [57] denotes the factorization of a matrix into a canonical form that is characterized by having non-zero elements (the eigenvalues) only on the main diagonal (note that not all matrices can be factorized this way, as discussed later). Let $\boldsymbol{A} \in \mathbb{R}^p$ be a diagonalizable square matrix. The eigendecomposition yields the equation $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{\Lambda}\boldsymbol{S}^{-1}$, where $\lambda_i = \boldsymbol{\Lambda}_{ii}$ are the eigenvalues of $\boldsymbol{A}$ and $\boldsymbol{S}_{*i}$ their corresponding eigenvectors, sharing the known property $\boldsymbol{A}\boldsymbol{S}_{*i} = \lambda_i \boldsymbol{S}_{*i}$. When a square matrix is not diagonalizable because of repeated eigenvalues, it can be factored into what is known as the Jordan Form, which in addition to the eigenvalues of the matrix, may contain unitary values in the immediate upper diagonal in the case of duplicate eigenvalues: $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}$, where

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_1 & & \\ & \ddots & \\ & & \boldsymbol{J}_r \end{bmatrix} \text{ and } \boldsymbol{J}_{s \in [1,\dots,r]} = \begin{bmatrix} \lambda_s & 1 & \dots & 0 \\ 0 & \lambda_s & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & \lambda_s \end{bmatrix}. \tag{5}$$

In the case of the Jordan Form, the eigenvectors corresponding to repeated eigenvalues are called generalized eigenvectors and have the property that $(\boldsymbol{A} - \lambda_s \boldsymbol{I})^j \boldsymbol{v}_j = 0$, where $\boldsymbol{v}_j$ is the j^{th} generalized eigenvector related to eigenvalue λ_s.

Finally, the eigenvalues of a real matrix may be complex numbers. This is inconvenient in the ensuing analysis, so rather than using complex arithmetic on these numbers, we choose a different representation over the so-called pseudo-eigenspace. Pseudo-eigendecomposition relies on the observation that complex eigenvalues of a real matrix always come in conjugate pairs. Relaxing the restriction of non-zero values on the main diagonal to include the immediate off-diagonal terms, we leverage the following equivalence:

$$\begin{aligned} \begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_{i+1} \end{bmatrix} \begin{bmatrix} \lambda_i & 0 \\ 0 & \lambda_{i+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_{i+1} \end{bmatrix}^{-1} &= \begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_i^* \end{bmatrix} \begin{bmatrix} re^{\theta i} & 0 \\ 0 & re^{-\theta i} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_i^* \end{bmatrix}^{-1} \\ &= \begin{bmatrix} re(\boldsymbol{v}_i) & im(\boldsymbol{v}_i) \end{bmatrix} \begin{bmatrix} r\cos(\theta) & r\sin(\theta) \\ -r\sin(\theta) & r\cos(\theta) \end{bmatrix} \begin{bmatrix} re(\boldsymbol{v}_i) & im(\boldsymbol{v}_i) \end{bmatrix}^{-1}, \end{aligned} \tag{6}$$

where $re(\boldsymbol{v})$ and $im(\boldsymbol{v})$ are the real and imaginary part of $\boldsymbol{v}$, respectively. In the case of a non-diagonal Jordan form, the columns are rearranged first (including the upper diagonal ones), and the conversion above is then performed. This representation is also called the Real Jordan Form.

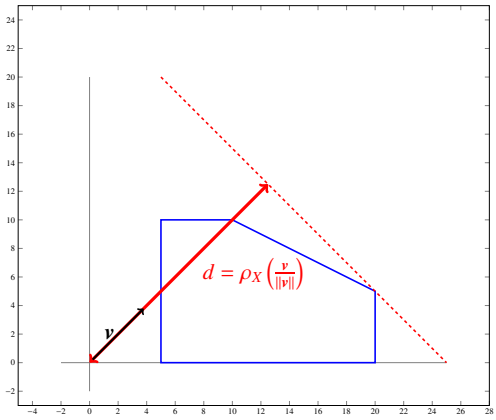

Fig. 1 Support function for a polyhedral set in $\mathbb{R}^2$. The distance between the tangent line and the origin is $\rho_X\left(\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}\right)$.

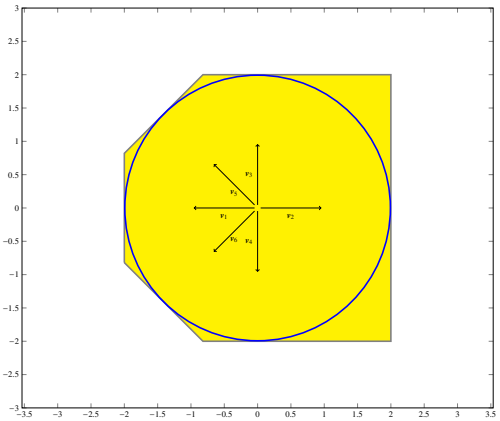

Fig. 2 Support function for a circular set using six directions ($\boldsymbol{v}_1 \cdots \boldsymbol{v}_6$). The resulting polyhedron is an over-approximation of the original set (note that directions need not be symmetrical).

2.4 Support functions

2.4.1 Definition of support functions

A support function is a convex function defined over the vector space $\mathbb{R}^p$, which describes the distance of a supporting hyperplane to the origin from a given set in $\mathbb{R}^p$, as shown in Figure 1.

Support functions can be used to describe a set by defining the distance of its convex hull with respect to the origin, given a number of directions. More specifically, a support function characterises the distance from the origin to the hyperplane that is orthogonal to the given direction and that touches its convex hull at its farthest. For example, the support function of a sphere centered at the origin given any unit vector $\boldsymbol{v}$ in $\mathbb{R}^3$, evaluates as the radius of the sphere.

The intersection of multiple half spaces, each obtained by sampling a support function in a specific direction, can generate a polyhedron (see Figure 2), as further discussed in the next section. Finitely sampled support functions (i.e. using a limited number of directions) are template polyhedra in which the directions are not fixed, which helps avoiding wrapping effects (wherein sampling in given directions creates an over-approximation of a set that is not aligned with said directions). The larger the number of distinct directions provided, the more precisely represented the set is. In more detail, given a direction $\boldsymbol{v} \in \mathbb{R}^p$, the support function of a non-empty set $X \subseteq \mathbb{R}^p$ in the direction of $\boldsymbol{v}$ is defined as

$$\rho_X : \mathbb{R}^p \to \mathbb{R}, \quad \rho_X(\boldsymbol{v}) = \sup\{\boldsymbol{x} \cdot \boldsymbol{v} : \boldsymbol{x} \in X\},$$

where $\boldsymbol{x} \cdot \boldsymbol{v} = \sum_{i=0}^{p} \boldsymbol{x}_i \boldsymbol{v}_i$ is the dot product of the two vectors. Support functions apply to any non-empty set $X \subseteq \mathbb{R}^p$, but they are most useful representing convex sets. We will restrict ourselves to the use of convex polyhedra, in which case the support function definition translates to solving the linear program

$$\rho_X(\boldsymbol{v}) = \max\{\boldsymbol{x} \cdot \boldsymbol{v} : \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d}\} .$$

2.4.2 Properties of support functions

Several properties of support functions allow us to reduce the complexity of operations. The most significant ones are [33]:

$$\begin{aligned}
&\rho_{kX}(\boldsymbol{v}) = \rho_X(k\boldsymbol{v}) = k\rho_X(\boldsymbol{v}) \text{ for } k \geq 0,\\
&\rho_{AX}(\boldsymbol{v}) = \rho_X(A^\intercal \boldsymbol{v}) \text{ where } A \in \mathbb{R}^{p\times p},\\
&\rho_{X_1 \oplus X_2}(\boldsymbol{v}) = \rho_{X_1}(\boldsymbol{v}) + \rho_{X_2}(\boldsymbol{v}),\\
&\rho_X(\boldsymbol{v}_1 + \boldsymbol{v}_2) \leq \rho_X(\boldsymbol{v}_1) + \rho_X(\boldsymbol{v}_2),\\
&\rho_{conv(X_1 \cup X_2)}(\boldsymbol{v}) = \max\{\rho_{X_1}(\boldsymbol{v}), \rho_{X_2}(\boldsymbol{v})\},\\
&\rho_{X_1 \cap X_2}(\boldsymbol{v}) \leq \min\{\rho_{X_1}(\boldsymbol{v}), \rho_{X_2}(\boldsymbol{v})\},
\end{aligned}$$

where $\boldsymbol{v}, \boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^p$. As can be seen by their structure, some of these properties reduce complexity to lower-order polynomial or even to constant time, by turning matrix-matrix multiplications ($O(p^3)$) into matrix-vector ($O(p^2)$), or into scalar ($O(p)$) multiplications.

2.4.3 Support functions in complex spaces

The literature does not state, to the best of our knowledge, any use of support functions in complex spaces. Since we are applying their concept to eigenspaces, which may have complex conjugate eigenvalues, we extend the definition of support functions to encompass corresponding operation on complex spaces, which are explicitly shown.

Theorem 1 *A support function in a complex vector field is a transformation:*

$$\rho_X(\boldsymbol{v}) : \mathbb{C}^p \to \mathbb{R} = \sup\{re(\boldsymbol{x} \cdot \boldsymbol{v}) : \boldsymbol{x} \in X \subseteq \mathbb{C}^p, \boldsymbol{v} \in \mathbb{C}^p\},$$

where $re(\cdot)$ defines the real part of a complex number. The dot product used here is commonly defined in a complex space as:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=0}^{p} a_i b_i^* , \quad \boldsymbol{a}, \boldsymbol{b} \in \mathbb{C}^p,$$

where the element b_i^* is the complex conjugate of b_i.

Proof Let $f : \mathbb{C}^p \to \mathbb{R}^{2p}$ and

$$\boldsymbol{x}' = f(\boldsymbol{x}) = \begin{bmatrix} re(\boldsymbol{x}) \\ im(\boldsymbol{x}) \end{bmatrix}, \quad \boldsymbol{v}' = f(\boldsymbol{v}^*) = \begin{bmatrix} re(\boldsymbol{v}) \\ -im(\boldsymbol{v}) \end{bmatrix}.$$

Using abstract interpretation [22] we define a Galois connection $\alpha(\boldsymbol{x}) = f(\boldsymbol{x})$ and $\gamma(\boldsymbol{x}') = f^{-1}(\boldsymbol{x}')$, which is clearly a one-to-one relation. We can therefore establish an equivalence between $\rho_X(\boldsymbol{v}) = \rho_{X'}(\boldsymbol{v}')$. □

As we shall see later, the imaginary part of the result of the dot product is not relevant to the support function, therefore we ignore it. Using support functions properties, we now have:

$$\rho_X(re^{i\theta}\boldsymbol{v}) = r\rho_X(e^{i\theta}\boldsymbol{v}),$$

which is consistent with the real case when $\theta = 0$. The reason why $e^{i\theta}$ cannot be factored out as a constant is because it executes a rotation on the vector, and therefore it follows the same rules as a matrix multiplication, namely

$$\rho_X(e^{i\theta}\boldsymbol{v}) \triangleq \rho_X(\boldsymbol{v}_2), \text{ where}$$
$$\begin{bmatrix} re(\boldsymbol{v}_2) \\ im(\boldsymbol{v}_2) \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} re(\boldsymbol{v}) \\ im(\boldsymbol{v}) \end{bmatrix}.$$

Notice the resemblance of this matrix to a pseudo-eigenvalue representation (See Equation (6)). Since the vectors we are interested in are conjugate pairs (because they have been created by a transformation into a matrix eigenspace), we can transform our problem into a pseudo-eigenvalue representation. Since this removes the imaginary part from the setup, we can evaluate the support function using the standard methods and properties by transforming the conjugate pairs into separate vectors representing the real and imaginary parts and rotation matrices as in the equation above.

Recently, [1] has presented a calculus for complex zonotopes that exploits at its core the same idea, namely using complex eigenvalues to represent ellipsoidal and linear boundaries for reach sets. Their projections into real space correspond to our semi-spherical approximations for variable inputs. Apart from the basic difference in domain (zonotopes vs polyhedra), which in itself changes some aspects of the problem, the main difference is that the authors perform their analysis over the complex space, whereas we ultimately work the idea in the real space by using pseudo-eigenspaces.

2.5 Convex polyhedra

A polyhedron is a subset of $\mathbb{R}^p$ with planar faces. Each face is supported by a hyperplane that creates a half-space, and the intersections of these hyperplanes are the edges (and vertices) of the polyhedron. A polyhedron is said to be convex if a line segment joining any two points of its surface is contained in its interior. Convex polyhedra are better suited than general polyhedra to define an abstract domain, mainly because they have a simpler representation and because operations over convex polyhedra are in general easier than over general polyhedra. There are a number of properties of convex polyhedra that make them ideal for abstract interpretation over continuous spaces, including their ability to reduce an uncountable set of real points into a countable set of faces, edges and vertices. Convex polyhedra retain their convexity across linear transformations, and are functional across a number of operations because they have a dual representation [31], as detailed next. The algorithm to switch between these two representations is given in Section 2.6.5.

2.5.1 Vertex representation

Since every edge in the polyhedron corresponds to a line between two vertices and every face corresponds to the area enclosed by a set of co-planar edges, a full description of the polyhedron is obtained by simply listing its vertices. Since linear operations retain the topological properties of the polyhedron, performing these operations on the vertices is sufficient to obtain a complete description of the transformed polyhedron (defined by the transformed vertices). Formally, a polyhedron can be described as a set $V \in \mathbb{R}^p$ such that $\boldsymbol{v} \in V$ is a vertex of the polyhedron.

2.5.2 Inequality representation (a.k.a. face representation)

The dual of the Vertex representation is the inequality representation, where each inequality represents a face of the polyhedron. Each face corresponds to a bounding hyperplane of the polyhedron (with the edges being

the intersection of two hyperplanes and the vertices being the intersection of p or more hyperplanes), and is described mathematically as a function of the vector that is normal to the hyperplane. This representation can be made minimal by eliminating redundant inequalities that do not correspond to any face of the polyhedron. If we examine this description closely, we can see that it corresponds to the support function of the vector normal to the hyperplane. Given this description we formalise the following: A convex polyhedron is a topological region in $\mathbb{R}^p$ described by the set

$$X = \{\boldsymbol{x} \in \mathbb{R}^p : \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d}, \boldsymbol{C} \in \mathbb{R}^{m \times p}, \boldsymbol{d} \in \mathbb{R}^m\}, \tag{7}$$

where the rows $\boldsymbol{C}_{i,*}$ for $i \in [1, \ldots, m]$ correspond to the transposed vectors normal to the faces of the polyhedron, and $\boldsymbol{d}_i$ for $i \in [1, \ldots, m]$ to the value of the support function of X in the corresponding direction. For simplicity in the presentation, we will extend the use of the support function operator as follows:

$$\rho'_X : \mathbb{R}^{m \times p} \to \mathbb{R}^m, \quad \rho'_X(\boldsymbol{M}) = \begin{bmatrix} \rho_X((\boldsymbol{M}_{\boldsymbol{1},*})^\intercal) \\ \rho_X((\boldsymbol{M}_{\boldsymbol{2},*})^\intercal) \\ \vdots \\ \rho_X((\boldsymbol{M}_{\boldsymbol{m},*})^\intercal) \end{bmatrix}.$$

2.6 Operations on convex polyhedra

There are a number of operations that we need to be able to perform on convex polyhedra.

2.6.1 Translations

Given a vertex representation V and a translation vector $\boldsymbol{t}$, the transformed polyhedron is

$$V' = \{\boldsymbol{v} + \boldsymbol{t} : \boldsymbol{v} \in V\}.$$

Given an inequality representation X and a translation vector $\boldsymbol{t}$, the transformed polyhedron is

$$X' = \{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d} + \boldsymbol{C}\boldsymbol{t}\}.$$

2.6.2 Linear transformations

Given a vertex representation V and a linear transformation $\boldsymbol{L}$, the transformed polyhedron is

$$V' = \boldsymbol{L}V.$$

Given an inequality representation X and a linear transformation $\boldsymbol{L}$, the transformed polyhedron corresponds to

$$X' \subseteq \{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{L}^+\boldsymbol{x} \leq \rho'_X((\boldsymbol{L}^+)^\intercal \boldsymbol{C}^\intercal)\},$$

where $\boldsymbol{L}^+$ represents the pseudo-inverse of $\boldsymbol{L}$ [53]. In the case when the inverse $\boldsymbol{L}^{-1}$ exists, then

$$X' = \left\{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{L}^{-1}\boldsymbol{x} \leq \boldsymbol{d}\right\}.$$

From this we can conclude that linear transformations are more efficient when using vertex representation, except when the inverse of the transformation exists and is know a-priori. This work makes use of this assumption to avoid alternating between representations.

2.6.3 Set sums

The addition of two polyhedra is such that the resulting set is that in which for all possible pairs of points inside the original polyhedra, the sum is contained in the result. This operation is commonly known as the Minkowski sum, namely

$$A \oplus B = \{a + b : a \in A, b \in B\}.$$

Given two vertex representations V_1 and V_2, the resulting polyhedron is

$$V = conv(V_1 \oplus V_2),$$

where $conv(\cdot)$ is the convex hull of the set of vertices contained in the Minkowski sum. Let

$$X_1 = \{\boldsymbol{x} : \boldsymbol{C}_1\boldsymbol{x} \leq \boldsymbol{d}_1\}, \quad X_2 = \{\boldsymbol{x} : \boldsymbol{C}_2\boldsymbol{x} \leq \boldsymbol{d}_2\},$$

be two sets, then

$$X_1 \oplus X_2 \subseteq X = \{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d}\},$$

where

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_2 \end{bmatrix}, \ \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_1 + \rho'_{X_2}(\boldsymbol{C}_1^{\intercal}) \\ \boldsymbol{d}_2 + \rho'_{X_1}(\boldsymbol{C}_2^{\intercal}) \end{bmatrix}.$$

Because these sets correspond to systems of inequalities, they may be reduced by removing redundant constraints. Note that if $\boldsymbol{C}_1 = \boldsymbol{C}_2$, then

$$X = X_1 \oplus X_2 = \{\boldsymbol{x} : \boldsymbol{C}_1\boldsymbol{x} \leq \boldsymbol{d}_1 + \boldsymbol{d}_2\}.$$

2.6.4 Set Hadamard products

Definition 1 Given two vertex representations V' and V'', we define the set Hadamard product operation using such representations as

$$V = V' \circ V'' = conv(\{\boldsymbol{v}' \circ \boldsymbol{v}'' : \boldsymbol{v}' \in V', \boldsymbol{v}'' \in V''\}),$$

where $\circ$ represents the Hadamard (coefficient-wise) product of the vectors.

Lemma 1 *The set $V = V' \circ V''$ introduced in the previous definition is a convex set containing all possible combinations of products between elements of sets V' and V''.*

Proof Given a convex set X with a vertex representation V, by definition we have

$$X' = \{t\boldsymbol{v}_1 + (1-t)\boldsymbol{v}_2, \boldsymbol{v}_1, \boldsymbol{v}_2 \in V, t \in [0,1]\} \subseteq X,$$

which extends to multiple points [13] as

$$X' = \left\{ \sum_{i=1}^{m} k_i \boldsymbol{v}_i, \sum_{i=1}^{m} k_i = 1, \boldsymbol{v}_i \in V, \text{ and } m = |V| \right\} \subseteq X.$$

Applying the Hadamard product, we obtain

$$X' = \{\boldsymbol{x}_1 \circ \boldsymbol{x}_2 : \boldsymbol{x}_1 \in X_1, \boldsymbol{x}_2 \in X_2\} \subseteq X = X_1 \circ X_2,$$

where $x_1 \circ x_2 = \sum_{i=1}^{|V'|} k_i \boldsymbol{v}'_i \circ \sum_{j=1}^{|V''|} k_j \boldsymbol{v}''_j$ with $\boldsymbol{v}'_i \in V', \boldsymbol{v}''_j \in V''$. Simplifying, we obtain

$$x_1 \circ x_2 = \sum_{i=1}^{|V'|} \sum_{j=1}^{|V''|} k_i \boldsymbol{v}'_i \circ k_j \boldsymbol{v}''_j = \sum_{i=1}^{|V'|} \sum_{j=1}^{|V''|} k_i k_j \boldsymbol{v}'_i \circ \boldsymbol{v}''_j = \sum_{ij=1}^{|V'||V''|} k_{ij} \boldsymbol{v}_{ij},$$

where $\boldsymbol{v}_{ij} = \boldsymbol{v}'_i \circ \boldsymbol{v}''_j \in V$ and $\sum_{ij=1}^{|V'||V''|} k_{ij} = \sum_{i=1}^{|V'|} \sum_{j=1}^{|V''|} k_i k_j = 1$. □

Note that in the case of the inequality representation, there is no direct result for this product. We must therefore enumerate the sets in one of the polyhedra, and use linear solving algorithms to find an over-approximation as

$$X \subseteq \{\boldsymbol{x} | \boldsymbol{x} \cdot \boldsymbol{t} < \max\{\rho_{X_2}(\boldsymbol{t} \circ \boldsymbol{v}') \mid \boldsymbol{v}' \in V'\}, \boldsymbol{t} \in T\}, \tag{8}$$

where $\boldsymbol{t}$ is a template direction for a face in the over-approximation, T is the set of directions selected for the over-approximation, and V' is the set of vertices of X.

2.6.5 Vertex enumeration

The vertex enumeration algorithm obtains a list of all vertices of a polyhedron, given a face representation of its bounding hyperplanes. Given the duality of the problem, it is also possible to find the bounding hyperplanes given a vertex description if the chosen algorithm exploits this duality. In this case the description of V is given in the form of a matrix inequality $\boldsymbol{V}\boldsymbol{x} \leq [\,1\ 1\ \cdots\ 1\,]^\intercal$ with $\boldsymbol{V} = [\,\boldsymbol{v}_1\ \cdots\ \boldsymbol{v}_m\,]^\intercal, \boldsymbol{v}_i \in V$. Similarly, $\boldsymbol{C}$ can be described as a set containing each of its rows. There are two algorithms that efficiently solve the vertex enumeration problem. lrs [4] is a reverse search algorithm, while cdd [31] follows the double description method. In this work we use the cdd algorithm for convenience in implementation (the original cdd was developed for floats, whereas lrs uses rationals). The techniques presented here can be applied to either. Let

$$C = \{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{x} \geq 0, \boldsymbol{C} \in \mathbb{R}^{n \times p}, \boldsymbol{x} \in \mathbb{R}^p\}.$$

be the polyhedral cone represented by $\boldsymbol{C}$. The pair $(\boldsymbol{C}, V)$ is said to be a double description pair if

$$C = \left\{\boldsymbol{\lambda}^\intercal V : V \in \mathbb{R}^p, \boldsymbol{\lambda} \in \mathbb{R}^{|V|}_{\geq 0}\right\},$$

where V is called the generator of X. Each element in V lies in the cone of X, and its minimal form (smallest m) has a one-to-one correspondence with the extreme rays of X if the cone is pointed (i.e., it has a vertex at the origin). This last can be ensured by translating a polyhedral description so that it includes the origin, and then translating the vertices back once they have been discovered (see Section 2.6).

We also point out that

$$\{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d}\} = \left\{\boldsymbol{x}' : [\,-\boldsymbol{C}\ \boldsymbol{d}\,]\,\boldsymbol{x}' \geq 0\right\}, \quad \text{where } \boldsymbol{x} \in \mathbb{R}^p \text{ and } \boldsymbol{x}' = \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \in \mathbb{R}^{p+1}.$$

The vertex enumeration algorithm starts by finding a base C_K which contains a number of vertices of the polyhedron. This can be done by pivoting over a number of different rows in $\boldsymbol{C}$ and selecting the feasible visited points, which are known to be vertices of the polyhedron (pivoting p times will ensure at least one vertex is visited if the polyhedron is non-empty). C_K is represented by $\boldsymbol{C}_K$ which contains the rows used for the pivots. The base C_K is then iteratively expanded to C_{K+i} by exploring the i^{th} row of $\boldsymbol{C}$ until $C_K = C$. The corresponding pairs $(\boldsymbol{C}_{K+i}, V_{K+i})$ are constructed using the information from $(\boldsymbol{C}_K, V_K)$ as follows.
Let $\boldsymbol{C}_K \in \mathbb{R}^{n_K \times p}$, $\boldsymbol{C}_{i,*} \in \mathbb{R}^{1 \times p}$, $V_K \in \mathbb{R}^p$,

$$H_i^+ = \{\boldsymbol{x} : \boldsymbol{C}_{i,*}\boldsymbol{x} > 0\}, \quad H_i^- = \{\boldsymbol{x} : \boldsymbol{C}_{i,*}\boldsymbol{x} < 0\}, \quad \text{and} \quad H_i^0 = \{\boldsymbol{x} : \boldsymbol{C}_{i,*}\boldsymbol{x} = 0\},$$

be the spaces outside, inside and on the i^{th} hyperplane and

$$V_K^+ = \{\boldsymbol{v}_j \in H_i^+\}, \quad V_K^- = \{\boldsymbol{v}_j \in H_i^-\}, \quad \text{and} \quad V_K^0 = \{\boldsymbol{v}_j \in H_i^0\},$$

the existing vertices lying on each of these spaces. Then [31],

$$V_{K+i} = V_K^+ \cup V_K^- \cup V_K^0 \cup V_K^i, \quad V_K^i = \{(\boldsymbol{C}_{i,*}\boldsymbol{v}^+)\boldsymbol{v}^- - (\boldsymbol{C}_{i,*}\boldsymbol{v}^-)\boldsymbol{v}^+ \mid \boldsymbol{v}^- \in V^-, \boldsymbol{v}^+ \in V^+\}.$$

3 Abstract Acceleration - Overview of the Algorithm

Abstract acceleration is a method that seeks to precisely describe the dynamics of a transition system over a number of steps using a concise description between the first and final steps. More precisely, it looks for a direct formula to express the post-image of an unfolded loop from its initial states. Formally, given the dynamics in equation (1) an acceleration formula aims at computing the reach tube based on (3) using a function f, such that $f(\cdot) = \tau^n(\cdot)$. In the case of models without inputs, this equation can be derived from the expression $\boldsymbol{x}_n = \boldsymbol{A}^n \boldsymbol{x}_0$.

3.1 Overview of the algorithm

The basic steps required to compute a reach tube using abstract acceleration are shown in Figure 3.

1. The process starts by performing eigendecomposition of the dynamics (based on matrix $\boldsymbol{A}$) in order to transform the problem into a simpler one. Since we use at this stage unsound arithmetics, the results are quantities identified by a tilde (as in $\tilde{\boldsymbol{S}}, \tilde{\boldsymbol{J}}$).
2. The second step involves upper-bounding the rounding errors in order to obtain sound results: bounds on eigenvalues for example are well known from the literature and can be obtained as $|\lambda - \tilde{\lambda}| < \|\tilde{\boldsymbol{S}}\tilde{\boldsymbol{J}}\tilde{\boldsymbol{S}}^{-1} - \boldsymbol{A}\|_2$. In general, a variety of off-the-shelf tools may be used, but since larger problems require numerical algorithms for scalability, all subsequent steps are performed using interval arithmetics in order to maintain soundness: we identify corresponding interval-based quantities with bold symbols (e,g, $\mathbb{S}, \mathbb{J}$), as well as subsequent matrix operations (e.g., computing the inverse of $\mathbb{S}$). Thus we obtain $\|\mathbb{S}\mathbb{J}\mathbb{S}^{-1} - \mathbb{A}\|_2$ by extending the original unsound matrices by one least significant bit element-wise. We also note that this equation is symmetric, which is why we use the estimated eigenvectors to calculate the error on the original eigenvalues (see [16] for further details). Whilst bounds on the eigenvectors can be calculated from the eigenvalues, we choose a more complex yet faster over-approximation, which is described in [16].
3. The inverse of the generalised eigenvectors ($\mathbb{S}^{-1}$) is calculated soundly (using interval arithmetics).
4. The problem is transformed into canonical form by multiplying both sides of the equation by $\mathbb{S}^{-1}$ (we use blackboard bold symbols to indicate interval vectors and matrices, which are employed to ensure sound operations), obtaining

$$X'_k = \mathbb{J}\left(X'_{k-1} \cap G'\right) + U', \quad \text{where} \quad X'_k = \mathbb{S}^{-1}X_k, \quad U' = \mathbb{S}^{-1}\boldsymbol{B}U, \quad G' = \{\boldsymbol{x} \mid \boldsymbol{G}\mathbb{S}\boldsymbol{x} \leq \boldsymbol{h}\}.$$

5. We calculate the number of iterations n based on the guard set, as explained in Section 7. If there are no guards, we set $n = \infty$. This number need not be exact: if we over-approximate the number of iterations, the resulting reach tube will further over-approximate the desired one.
6. We over-approximate the dynamics subject to general inputs (for parametric inputs or in the absence of inputs this step will be ignored), using the techniques described in Section 6.2.
7. We calculate the accelerated dynamics using the techniques described in Section 5.1.
8. We transform the input and initial sets into vertex representation, to be used as source for the reach tube calculation.
9. We employ a sound simplex algorithm [16] to evaluate the convex-set Hadamard product of the abstract dynamics and of the initial set. The two most important elements of the sound simplex are that it uses interval arithmetics to pivot, and that at every stage it verifies the intersection between vertices in order to avoid pivoting on unsound solutions. The latter step is better explained by considering that the simplex algorithm operates by visiting adjacent vertices. The interval arithmetics ensure that the solution at the last visited vertex is sound, but if there is an intersection, the new pivot may choose to start on the intersecting vertex instead (which is not sound), thus, by checking the intersection and extending the interval to encompass both vertices, we retain soundness (see [16] for details).

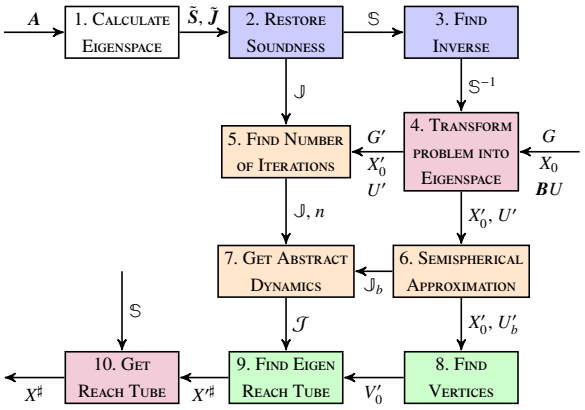

Fig. 3 Block diagram describing the different steps used to calculate the abstract reach tube of a model via Abstract Acceleration. The white box is the numerical eigensolver stage. Blue boxes are soundness restoration stages. Red boxes represent linear transformations of the problem. The orange boxes denote abstractions defined in this paper, and the green boxes the reachability computation in the abstract domain.

10. Since we have operated so far in the eigenspace, we transform the reach tube back into the state space via multiplication by $\mathbb{S}$.

4 Abstract Matrices in Abstract Acceleration

We introduce the concept of abstract matrix.

Definition 2 An abstract matrix $\mathcal{A}^n \subseteq \mathbb{R}^{p\times p}$ is an over-approximation of the union of the powers of the matrix $\boldsymbol{A}^k$, such that $\mathcal{A}^n \supseteq \left\{\boldsymbol{A}^k : k \in [0, \dots, n]\right\}$. Its application to the initial set X_0 results in

$$\hat{X}_n^\sharp = \mathcal{A}^n X_0, \tag{9}$$

such that $\hat{X}_n^\sharp \supseteq \hat{X}_n$ is an over-approximation of the reach tube described in Equation (4).

Next we explain how to compute such abstract matrices. For simplicity, we first describe this computation for matrices $\boldsymbol{A}$ with real eigenvalues, whereas the extension to the complex case will be addressed in Section 4.1. Similar to [44], we first have to compute the Jordan normal form of $\boldsymbol{A}$. Let $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}$ where $\boldsymbol{J}$ is the normal Jordan form of $\boldsymbol{A}$, and $\boldsymbol{S}$ is made up by the corresponding generalized eigenvectors. We can then easily compute $\boldsymbol{A}^n = \boldsymbol{S}\boldsymbol{J}^n\boldsymbol{S}^{-1}$, where given a set of r eigenvalues λ_s with geometric multiplicities p_s and $s \in [1, \dots, r]$, we have

$$\boldsymbol{J}^n = \begin{bmatrix} \boldsymbol{J}_1^n & & \\ & \ddots & \\ & & \boldsymbol{J}_r^n \end{bmatrix}, \quad \text{where} \quad \boldsymbol{J}_s^n = \begin{bmatrix} \lambda_s^n & \binom{n}{1}\lambda_s^{n-1} & \cdots & \binom{n}{p_s-1}\lambda_s^{n-p_s+1} \\ & \lambda_s^n & \binom{n}{1}\lambda_s^{n-1} & \vdots \\ \vdots & & \ddots & \vdots \\ & & & \lambda_s^n \end{bmatrix}. \tag{10}$$

The abstract matrix $\mathcal{A}^n$ is computed as an abstraction over a set of vectors $\boldsymbol{m}^k \in \mathbb{R}^p, k \in [1, \dots, n]$ of distinct entries of $\boldsymbol{J}^k$, as explained below.
Let $\boldsymbol{I}_s = [\,1\ 0\ \cdots\ 0\,] \in \mathbb{R}^{p_s}$. The vector $\boldsymbol{m}^k$ is obtained by the transformation φ^{-1} (which is always invertible) as

$$\boldsymbol{m}^k = \varphi^{-1}\left(\boldsymbol{J}^k\right) = \left[\, \boldsymbol{I}_1\boldsymbol{J}_1^k \ \cdots\ \boldsymbol{I}_r\boldsymbol{J}_r^k \,\right]^\intercal \in \mathbb{R}^p, \tag{11}$$

such that $\boldsymbol{J}^k = \varphi(\boldsymbol{m}^k)$.

If $\boldsymbol{J}$ is diagonal [44], then $\boldsymbol{m}^k$ results in the vector made up of powers of the eigenvalues, $[\ \lambda_1^k \ \cdots \ \lambda_p^k\ ]$. The diagonal entries in the abstract matrix is thus bound by the intervals

$$\begin{cases} \left[\min\{|\lambda_s|^0, |\lambda_s|^n\}, \max\{|\lambda_s|^0, |\lambda_s|^n\}\right], & \lambda_s \in \mathbb{R}^+ \\ \left[-\max\{|\lambda_s|^0, |\lambda_s|^n\}, \max\{|\lambda_s|^0, |\lambda_s|^n\}\right], & \text{otherwise} \end{cases}, \text{ where } s \in [1, \dots, r], r = p. \tag{12}$$

We observe that the spectrum of the abstract matrix $\sigma(\mathcal{A}^n)$, which can be derived from its entries in Equation (12), over-approximates $\bigcup_{k\in[1,\dots,n]} \sigma(\boldsymbol{A}^k)$.

In the case of the s^{th} Jordan block $\boldsymbol{J}_s$ with geometric multiplicity $p_s > 1$, observe that the first row of $\boldsymbol{J}_s^n$ contains all (possibly) distinct entries of $\boldsymbol{J}_s^n$. Hence, the vector section $\boldsymbol{m}_s$ is the concatenation of the (transposed) first row vectors $\left(\lambda_s^n, \binom{n}{1}\lambda_s^{n-1}, \dots, \binom{n}{p_s-1}\lambda_s^{n-p_s+1}\right)^\intercal$ of $\boldsymbol{J}_s^n$.

Since φ transforms the vector $\boldsymbol{m}$ into the shape of (10) of $\boldsymbol{J}^n$, it is called a *matrix shape* [44]. We then define the abstract matrix as

$$\mathcal{A}^n = \{\boldsymbol{S}\ \varphi(\boldsymbol{m})\ \boldsymbol{S}^{-1} : \boldsymbol{\Phi m} \leq \boldsymbol{f}\}\ , \tag{13}$$

where the constraint $\boldsymbol{\Phi m} \leq \boldsymbol{f}$ is synthesised from intervals associated to the individual eigenvalues and to their combinations. More precisely, we compute polyhedral relations: for any pair of eigenvalues (or distinct entries) within $\boldsymbol{J}$, we find an over-approximation of the convex hull containing the points

$$\left\{\boldsymbol{m}^k : k \in [1, \dots, n]\right\} \subseteq \{\boldsymbol{m} : \boldsymbol{\Phi m} \leq \boldsymbol{f}\}\,.$$

The reason for evaluating the convex hull over pairs of points is twofold. In the first instance, we note that the set $\left\{\boldsymbol{m}^k \mid k \in [1, \dots, n]\right\}$ is in general not convex. This makes it hard to find its support in arbitrary directions. Ideal directions would be the normals to the gradients of the function, namely ∇m^k, which would provide the tightest over-approximation at iteration k. However, as will be seen below, when combining negative or complex conjugate eigenvalues, the corresponding hyperplane tangent may intersect the set, and thus it cannot be used to define its convex hull. The second reason for choosing pairwise directions is practical: we need an even distribution of the directions in $\mathbb{R}^p$, and it is easier to do this in a pairwise manner. More elaborate search of a set of directions might leverage convex optimization techniques, which however come with a computational overhead: as such, they fall outside the scope of this work.

4.1 Abstract matrices in complex spaces

To deal with complex numbers in eigenvalues and eigenvectors, [44] employs the real Jordan form for conjugate eigenvalues $\lambda = re^{i\theta}$ and $\lambda^* = re^{-i\theta}$ ($\theta \in [0, \pi]$), so that

$$\begin{bmatrix} \lambda & 0 \\ 0 & \lambda^* \end{bmatrix} \quad \text{is replaced by} \quad r\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

Although this equivalence will be of use once we evaluate the progression of the model, calculating powers under this notation is often more difficult than handling directly the original matrices with complex values.

In the case of real eigenvalues we have abstracted the entries in the power matrix $\boldsymbol{J}_s^n$ by ranges of eigenvalues $[\min\{\lambda_s^0 \cdots \lambda_s^n\}, \max\{\lambda_s^0 \cdots \lambda_s^n\}]$ forming a hypercube. In the complex case, where the rotations describe spherical behaviour, we can do something similar by rewriting eigenvalues into polar form $\lambda_s = r_s e^{i\theta_s}$ and enclosing the radius in the interval $[0, \bar{r}_s]$, where $\bar{r}_s = \max\{r_s^k : k \in [0, \dots, n]\}$ (in the worst case scenario this is over-approximated by a hyper-box with $\lambda_s^k \in [-\bar{r}_s, \bar{r}_s] + [-\bar{r}_s, \bar{r}_s]i$, but we will introduce tighter bounds in the course of this work).

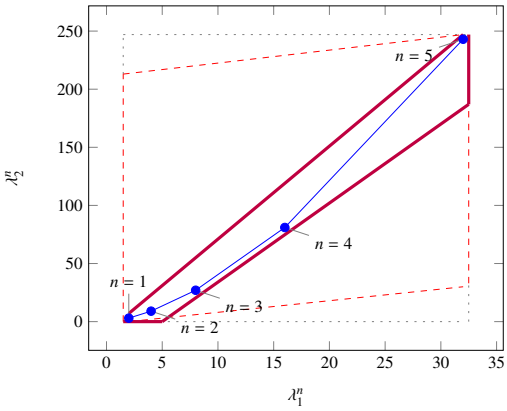

Fig. 4 Polyhedral faces over $\mathbb{R}^2$ for pairs of eigenvalues $(\lambda_1^n, \lambda_2^n)$ where λ_1=2, λ_2=3, and $1 \leq n \leq 5$. Bold purple lines represent supports. The dotted grey and dashed red polytopes show logahedral approximations (box and octagon) used in [44]. Note the scales (the sloped dashed lines are parallel to the x=y line, and the dashed red polytope hides two small faces yielding an octagon).

5 Abstract Acceleration without Inputs

5.1 Using support functions for abstract acceleration

As an improvement over [44], the rows in $\boldsymbol{\Phi}$ and $\boldsymbol{f}$ (see (13)) can be obtained by refined sampling of the support functions of these sets. The choice of directions for these support functions results in an improvement over the logahedral abstractions used in previous work [36, 37, 44] (see Figures 4 - 7). This approach works thanks to the convex properties of the exponential progression. We consider five cases:

1. Positive real eigenvalues
   The exponential curve is cut along the diagonal between the eigenvalues with maximum and minimum range to create a supporting hyperplane. A third point taken from the curve is used to test the direction of the corresponding template vector. An arbitrary number of additional supporting hyperplanes are created by selecting pairs of adjacent points in the curve and creating the corresponding support functions, as shown in Figure 4.
2. Complex conjugate eigenvalue pairs
   In the case of complex conjugate pairs, the eigenvalue map corresponds to a logarithmic spiral (See Figure 5). In this case, we must first extract the number of iterations (denoted by $\overline{k}$) required for a full cycle. For convergent eigenvalues ($|\lambda| < 1$), only the first $\overline{k}$ iterations have an effect on the support functions, while in the divergent case only the last $\overline{k}$ iterations are considered (since symmetrically, it corresponds to the reverse spiral case). Support functions are found for adjacent pairs, checking the location of the first point for convergent eigenvalues, and that of the last point for divergent eigenvalues. If a point falls outside of the supporting half-space, we look for an interpolant point that closes the spiral and that is tangent to the origin. This last check is performed as a binary search over the remaining points in the circle (noting that the supporting planes would exclude the considered point) to achieve maximum tightness (see Figure 5). Further consideration is taken to ensure that subsequent iterations fall within the envelope found on the first/last rotation. This is ensured by extending the support functions outwards by a factor $f = \max\left(\{1, |\lambda|^{\hat{n}} \cos(\theta)^{-1}\}\right)$, where θ is the angle of the eigenvalue pair and $\hat{n} = n$ for the convergent case or $\hat{n} = \frac{1}{n}$ for the divergent case. When this value is too large, we use an interpolation to find better supports. This is achieved by finding a pair such that the first point is obtained from λ^k and the second from $(\lambda^{\frac{1}{m}})^{mk+1}$. The relaxation factor then becomes $\cos\left(\frac{\theta}{m}\right)^{-1}$.

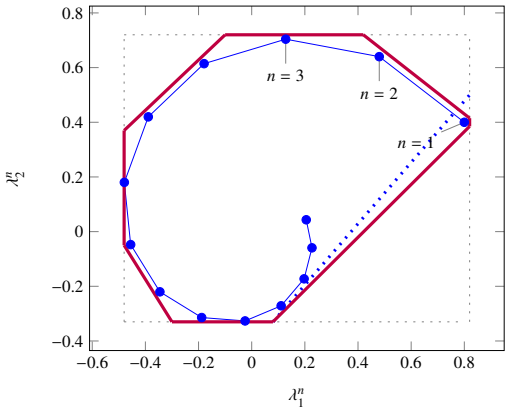

Fig. 5 Polyhedral faces projected onto $\mathbb{R}^2$ for complex conjugate eigenvalues $(\lambda_1^n, \lambda_2^n)$ where $\lambda_1{=}0.8+0.4i$, $\lambda_2{=}0.8-0.4i$, and $1{\leq}n{\leq}14$. Bold purple lines represent supports. The blue dotted line shows the supporting hyperplane that excludes the point obtained with $n = 1$, which is replaced by a supporting hyperplane tangent to the spiral but touching said origin.

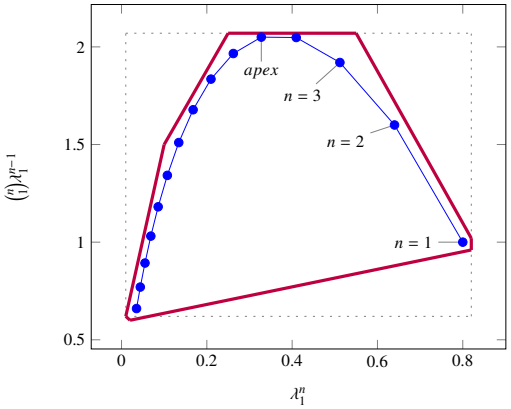

Fig. 6 Polyhedral faces in $\mathbb{R}^2$ related to a Jordan block $(\lambda_1^n, \binom{n}{1}\lambda_1^{n-1})$, where $\lambda_1{=}0.8$ and $1{\leq}n{\leq}15$. Bold purple lines represent supports found in this work.

3. Equal eigenvalues
   When two eigenvalues are equivalent, the resulting support functions are those orthogonal to the $x = y$ plane, intersecting the square created by the maximum and minimum values[1].
4. Jordan blocks of non-trivial size (> 1)
   In the case of eigenvalues with geometric multiplicities, we find three shapes. When both elements in the pair are convergent, since the convex sets can be "sharp", it is important to find the apex of the upper diagonals in order to minimise the over-approximation (see Figure 6). When both elements are divergent, the shape is similar to a positive valued pair since there is no extremum. Finally, when comparing different Jordan blocks, one convergent and one divergent, we evaluate the containing hyperbox, thus avoiding the change in convexity at the apex.
5. Negative eigenvalues and combinations of real eigenvalues with conjugate pairs
   When comparing a positive real eigenvalue to a complex conjugate or a negative one, we must account for the changes of sign in the progression of the latter. We compute envelopes of the progression of the corresponding dynamics, which are obtained via convex over-approximations (cf. Figure 7). In the case of complex eigenvalues, we use the absolute value in order to determine the envelope. If both eigenvalues

[1] As an alternative, a space reduction method can be applied to remove one of the eigenspaces, and the original space can be restored after the reduced reach tube has been evaluated.

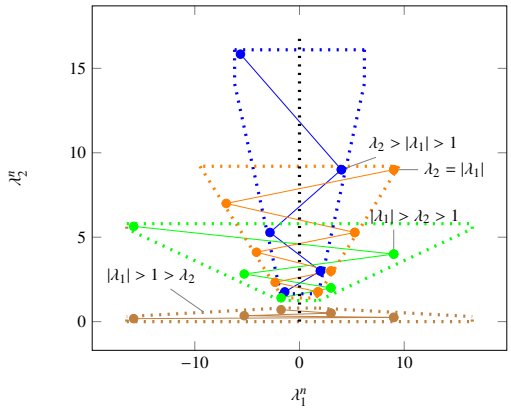

Fig. 7 Polyhedral faces over $\mathbb{R}^2$, of different eigenvalue ratios (note that the curves obtained from the progression of the blue and orange dots are convex w.r.t. the λ_2^n-axis, whereas the green and brown are concave). Dotted lines represent convex supports for these layouts.

have rotating dynamics, we would require full symmetry along the four quadrants, thus we end up with a hyper-box with vertices at the farthest points from the origin.

An additional drawback of [44] is that calculating the exact Jordan form of any matrix is computationally expensive for large-dimensional matrices. We will instead leverage numerical algorithms that provide an approximation of the Jordan normal form and soundly account for the associated numerical errors. We use properties of eigenvalues to relax $\boldsymbol{f}$ by finding the maximum error in the calculations, which can be determined by computing the norm $\delta_{max} = \|\hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}}^{-1} - \mathbb{A}\|$, where $\hat{\mathbb{J}}$ and $\hat{\mathbb{S}}$ are the eigenvalues and eigenvectors of $\boldsymbol{A}$ calculated numerically [16]. Let us recall that the notation above is used to represent interval matrices, and that all operations are performed using interval arithmetics with outward rounding in order to ensure soundness. The constraints in $\boldsymbol{\Phi m} \leq \boldsymbol{f}$ are then computed by considering the ranges of eigenvalues $|\lambda_s \pm \delta_{max}|^k$, which are represented in Figure 4 with blue circles.

The outward relaxation of the support functions ($\boldsymbol{f}$) follows a principle similar to that in [32], and reduces the tightness of the over-approximation, while ensuring the soundness of the obtained abstract matrix $\mathcal{A}^n$. Graphically, this is equivalent to moving the faces of a polyhedron outward, which has practically a minimal impact due to the small magnitude of δ_{max}. It is also worth noting that the transformation matrices into and from the eigenspace will also introduce over-approximations due to the intervals, and will exacerbate the over-approximations due to the condition number related to the eigenvalues.

One can still use exact arithmetics with a noticeable improvement over previous work; however, for larger-dimensional models the option of using floating-point arithmetic, while taking into account errors and meticulously setting rounding modes, provides a 100-fold plus improvement, which can make a difference towards rendering verification practically feasible. For a full description on the numerical techniques described here see [16].

5.2 Abstract matrices and support functions

Since we are describing operations using abstract matrices and support functions, we briefly review the nature of these operations and the properties that support functions retain within this domain. Let $X \in \mathbb{R}^p$ be a set and $\mathcal{A} \in \mathcal{R}^{p\times p}$ an abstract matrix for the same space. From Equation (13) we have

$$\mathcal{A} = \bigcup \boldsymbol{S}\varphi(\boldsymbol{m})\boldsymbol{S}^{-1}, \text{ where } \boldsymbol{\Phi m} \leq \boldsymbol{f},$$

which leads to

$$\rho_{\mathcal{A}X}(\boldsymbol{v}) = \rho_{\boldsymbol{S}\varphi(\boldsymbol{m})\boldsymbol{S}^{-1}X}(\boldsymbol{v}) = \rho_{\varphi(\boldsymbol{m})\boldsymbol{S}^{-1}X}(\boldsymbol{S}^\intercal \boldsymbol{v}), \tag{14}$$

where $\rho_{\varphi X}(\boldsymbol{v}) = \sup\left\{\rho_\varphi(\boldsymbol{x} \circ \boldsymbol{v}) : \boldsymbol{x} \in X\right\}$, and $\rho_\varphi(\boldsymbol{v}) = \sup\{\boldsymbol{m} \cdot \varphi^{-1}(\boldsymbol{v}) : \boldsymbol{\Phi m} \leq \boldsymbol{f}\}$. Here, $\boldsymbol{x} \circ \boldsymbol{y}$ is the Hadamard product, where $(\boldsymbol{x} \circ \boldsymbol{y})_i = \boldsymbol{x}_i \boldsymbol{y}_i$, and $\varphi^{-1}(\cdot)$ is the inverse operation of $\varphi(\cdot)$. We also define

$$\begin{aligned}
\rho_{\mathcal{A}X}(\boldsymbol{v}) &= \sup\left\{\rho_{\boldsymbol{a}X}(\boldsymbol{v}),\ \boldsymbol{a} \in \mathcal{A}\right\} \\
&= \sup\left\{\boldsymbol{S}\varphi(\boldsymbol{m})\boldsymbol{S}^{-1}\boldsymbol{x} \cdot \boldsymbol{v},\ \boldsymbol{x} \in X, \boldsymbol{\Phi m} \leq \boldsymbol{f}\right\} \\
&= \sup\left\{\varphi(\boldsymbol{m})\boldsymbol{S}^{-1}\boldsymbol{x} \cdot \boldsymbol{S}^\intercal \boldsymbol{v},\ \boldsymbol{x} \in X, \boldsymbol{\Phi m} \leq \boldsymbol{f}\right\} \\
&= \sup\left\{\rho_\varphi(\boldsymbol{S}^{-1}\boldsymbol{x} \circ \boldsymbol{S}^\intercal \boldsymbol{v}),\ \boldsymbol{x} \in X\right\},
\end{aligned}$$

and, in order to simplify the nomenclature, we write

$$\rho_{\mathcal{A}X}(\boldsymbol{v}) = \rho_X(\mathcal{A}^\intercal \boldsymbol{v}). \tag{15}$$

6 General Abstract Acceleration with Inputs

6.1 Acceleration of parametric inputs

Let us now consider the following over-approximation for τ on sets:

$$\tau^\sharp(X_0, U) = \boldsymbol{A}X_0 \oplus \boldsymbol{B}U, \tag{16}$$

and add a restriction to constant (also called parametric) inputs, namely where $\boldsymbol{u}_k = \boldsymbol{u}_0, \forall k > 0$ and $\boldsymbol{u}_0 \in U$. Unfolding (3) (ignoring the presence of the guard set G for the time being), we obtain

$$X_n = \boldsymbol{A}^n X_0 \oplus \sum_{k=0}^{n-1} \boldsymbol{A}^k \boldsymbol{B}U. \tag{17}$$

We further simplify the sum $\sum_{k=0}^{n-1} \boldsymbol{A}^k \boldsymbol{B}U$, exploiting the following result from linear algebra.

Lemma 2 *If $\boldsymbol{I} - \boldsymbol{A}$ is invertible, then*

$$\sum_{k=0}^{n-1} \boldsymbol{A}^k = (\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1}.$$

If furthermore $\lim_{n\to\infty} \boldsymbol{A}^n = 0$, *then* $\lim_{n\to\infty} \sum_{k=0}^{n} \boldsymbol{A}^k = (\boldsymbol{I} - \boldsymbol{A})^{-1}$.

The inverse $(\boldsymbol{I} - \boldsymbol{A})^{-1}$ does not exist for eigenvalues equal to 1, i.e. we need $1 \notin \sigma(\boldsymbol{A})$, where $\sigma(A)$ is the spectrum (the set of all the eigenvalues) of matrix $\boldsymbol{A}$. In order to overcome this problem, we introduce the eigendecomposition of $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}$, (and trivially $\boldsymbol{I} = \boldsymbol{S}\boldsymbol{I}\boldsymbol{S}^{-1}$), and by the distributive and transitive properties we obtain

$$(\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1} = \boldsymbol{S}(\boldsymbol{I} - \boldsymbol{J}^n)(\boldsymbol{I} - \boldsymbol{J})^{-1}\boldsymbol{S}^{-1}\ .$$

Although $(\boldsymbol{I} - \boldsymbol{J})$ is stil not invertible, this representation allows us to accelerate the eigenvalues individually, trivially noticing that $\sum_{k=0}^{n-1} 1^k = n$ for unitary eigenvalues (thus eliminating the need to calculate said inverse for these eigenvalues). Using the properties above, and translating the problem into the generalised eigenspace to account for unit eigenvalues, we obtain the following representation:

$$(\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1} = \boldsymbol{S}\boldsymbol{D}^n\boldsymbol{S}^{-1}, \tag{18}$$

given

$$\boldsymbol{D}^n_{i,j} = \begin{cases} 0 & gm(\lambda_i) \leq k \vee n < k \\ d(\lambda_i, n, 0) & i = j \\ \binom{n+1}{k+1} & \lambda_i = 1 \\ d(\lambda_i, n, j-i) & \lambda_i \neq 1, \end{cases}$$

where

$$d(\lambda_i, n, 0) = \sum_{k=0}^{n-1} \lambda_i^k = \begin{cases} n & \lambda_i = 1 \\ \frac{1-\lambda_i^n}{1-\lambda_i} & \lambda_i \neq 1, \end{cases}$$
$$d(\lambda_i, n, k) = \frac{-1^k}{k+1} \frac{1-\lambda_i^n}{(1-\lambda_i)^{k+1}} + \sum_{j=1}^{k} \frac{-1^{k-j}}{k-j} \binom{n}{j-1} \frac{\lambda_i^{n-j-1}}{(1-\lambda_i)^{k-j}},$$

and where $gm(\cdot)$ denotes the geometric multiplicity of the given eigenvalue, and $k = j - i$. With these notions in hand, we next define the abstract acceleration of parametric inputs.

Theorem 2 *The abstract acceleration is defined as*

$$\hat{\tau}^{\sharp n}(X_0, U) =_{\text{def}} \mathcal{A}^n X_0 \oplus \mathcal{B}^n U, \tag{19}$$

where $\mathcal{B}^n \supseteq \bigcup_{k\in[1,\dots,n]} \boldsymbol{S}(\boldsymbol{D}^k)\boldsymbol{S}^{-1}\boldsymbol{B}$, is an over-approximation of the n-reach tube, namely $\hat{X}_n \subseteq \hat{\tau}^{\sharp n}(X_0, U)$.

Proof From Equation (4) we have

$$\hat{X}_n = \hat{\tau}^n(X_0, U) = \bigcup_{k\in[0,\dots,n]} \tau^k(X_0, U).$$

Using Equation (17), we expand this into

$$\hat{X}_n = \bigcup_{k\in[0,\dots,n]} \boldsymbol{A}^k X_0 \oplus \sum_{j=0}^{k-1} \boldsymbol{A}^j \boldsymbol{B} U \subseteq \bigcup_{k\in[0,\dots,n]} \boldsymbol{A}^k X_0 \oplus \bigcup_{k\in[0,\dots,n]} \sum_{j=0}^{k-1} \boldsymbol{A}^j \boldsymbol{B} U,$$

then replace

$$\hat{X}_n \subseteq \mathcal{A}^n X_0 \oplus \bigcup_{k\in[1,\dots,n]} \boldsymbol{S}(\boldsymbol{D}^k)\boldsymbol{S}^{-1}\boldsymbol{B}U$$

and finally obtain

$$\hat{X}_n \subseteq \hat{\tau}^{\sharp n}(X_0, U).$$

The quantities $\mathcal{A}^n$ and $\mathcal{B}^n$ are calculated using the techniques described in Section 5.1, where special consideration is taken to evaluate pairs comprising equal eigenvalues. Figure 8 shows an example of such a pair. Since both functions are monotonic, the set is convex. The techniques applied to positive real eigenvalues (see Section 5.1) therefore stands. □

6.2 Acceleration of time-varying inputs

In order to generalise the previous results to time-varying inputs, we will over-approximate the term $\boldsymbol{B}U$ over the eigenspace by a semi-spherical enclosure, namely a set where complex conjugate eigenvalues are enclosed by a spherical support with centre $\boldsymbol{u}'_c$ and the radius of U'_b, whereas real eigenvalues are enclosed by hyper-rectangles (dashed symbols represent elements in the eigenspace). To this end, we first rewrite

$$U'_J = \boldsymbol{S}^{-1}\boldsymbol{B}U = \{\boldsymbol{u}'_c\} \oplus U'_d,$$

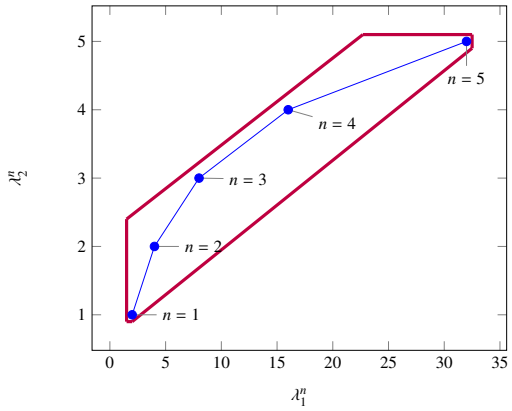

Fig. 8 Polyhedral faces (λ_1^n, n) over $\mathbb{R}^2$, where λ_1=2, λ_2=3, and 1≤n≤5. Bold purple lines represent supports found in this work.

where $\boldsymbol{u}'_c$ is the centre of the smallest hyperbox (interval hull) containing U'_J, and $U'_d = \{\boldsymbol{u} : \boldsymbol{u} + \boldsymbol{u}'_c \in U'_J\}$:

$$\boldsymbol{u}'_{ci} = \frac{1}{2}(\rho_{U'_J}(\boldsymbol{v_i}) + \rho_{U'_J}(-\boldsymbol{v_i})), \text{ where } \boldsymbol{v}_{ij} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}. \tag{20}$$

We then over-approximate U'_d via U'_b, by the maximum radius in the directions of the complex eigenvalues (cf. illustration in Figure 9). Let

$$\Lambda = \{\lambda_i : im(\lambda_i) \geq 0\}$$

be the set of eigenvalues of $\boldsymbol{A}$, where $im(\cdot)$ is the imaginary value of a complex number, and conjugate pairs are represented only by one member of the pair. Let us define the function $f_b : \mathbb{R}^p \to \mathbb{R}^{p_b}$, where p_b is the cardinality of Λ, such that

$$f_b(\boldsymbol{v}) = \text{red}(\boldsymbol{v}_b), \text{ where } (\boldsymbol{v}_b)_i = \begin{cases} 0 & \lambda_i \notin \Lambda \\ |\boldsymbol{v}_i| & \lambda_i \neq \lambda^*_{i+1} \\ \sqrt{\boldsymbol{v}_i^2 + \boldsymbol{v}_{i+1}^2} & \lambda_i = \lambda^*_{i+1} \end{cases},$$

$i \in [1, ..., p]$ and red$(\cdot)$ is a function that reduces the dimension of a vector by removing the elements where $\lambda_i \notin \Lambda$ (*i.e.* the null elements in $\boldsymbol{v}_b$, such that if for instance $\boldsymbol{v}_b = [\, v_1 \; 0 \; v_3 \; \dots \; v_p \,]^\intercal$, then red$(\boldsymbol{v}_b) = [\, v_1 \; v_3 \; \dots \; v_p \,]^\intercal$). Extending this to matrices we have

$$F_b : \mathbb{R}^{r \times p} \to \mathbb{R}^{r \times p_b}, \quad F_b(\boldsymbol{C}) = \boldsymbol{C}_b \text{ where } (\boldsymbol{C}_b)_{i,*} = f_b(\boldsymbol{C}_{i,*}),$$

where r denotes the number of inequalities describing a set in $\mathbb{R}^p$. Finally

$$\begin{aligned} &U'_d = \{\boldsymbol{u} \mid \boldsymbol{C}'_u \boldsymbol{u} \leq \boldsymbol{d}'_u\}, U'_d \subseteq U'_b, \text{ with} \\ &U'_b = \{\boldsymbol{u} \mid F_b(\boldsymbol{C}'_u) f_b(\boldsymbol{u}) \leq f_b(\boldsymbol{d}'_u)\}, \text{ and} \\ &\boldsymbol{B}U \subseteq U_b \oplus U_c, \text{ where } U_b = \boldsymbol{S}U'_b \text{ and } U_c = \{\boldsymbol{S}\boldsymbol{u}'_c\}. \end{aligned} \tag{21}$$

Since the description of U'_b is no longer polyhedral in $\mathbb{R}^p$, we will also create an over-approximation $\boldsymbol{J}_b$ of $\boldsymbol{J}$ in the directions of the complex eigenvectors, in a similar way as we generated U'_b for U'_d . More precisely,

$$\boldsymbol{J}_b = \begin{bmatrix} \boldsymbol{J}_{b1} & & \\ & \ddots & \\ & & \boldsymbol{J}^n_{br} \end{bmatrix}, \quad \text{where } \forall s \in [1, \dots, r] \begin{cases} \lambda_j \in \boldsymbol{J}_{bs} = |\lambda_i| \in \boldsymbol{J}_s \cap \Lambda \\ gm(\boldsymbol{J}_{bs}) = gm(\boldsymbol{J}_s), \end{cases} \tag{22}$$

and $gm(\cdot)$ is the geometric multiplicity of the specific Jordan block.

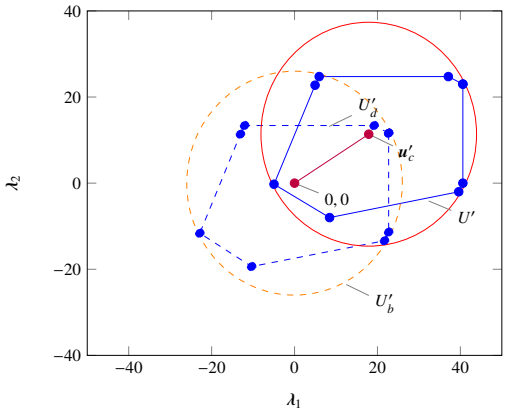

Fig. 9 Relaxation of an input set within a complex subspace, in order to make it invariant to matrix rotations. Dashed lines and curves denote translated quantities onto the origin.

Definition 3 Given a matrix $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}$ and a vector $\boldsymbol{x}$, we define the following operations:

$$F_b^*(\boldsymbol{A}, \boldsymbol{x}) = \boldsymbol{S} f_b^{-1}\left(F_b(\boldsymbol{J}) f_b(\boldsymbol{S}^{-1}\boldsymbol{x})\right). \tag{23}$$

Finally, we refer to the accelerated sets

$$U_b^n = \left\{F_b^*((\boldsymbol{I}-\boldsymbol{A}^n), F_b^*((\boldsymbol{I}-\boldsymbol{A})^{-1}, \boldsymbol{u})) \mid \boldsymbol{u} \in U_b\right\}, \quad U_c^n = (\boldsymbol{I}-\boldsymbol{A}^n)(\boldsymbol{I}-\boldsymbol{A})^{-1}U_c, \quad U_{cb}^n = U_c^n \oplus U_b^n.$$

Returning to our original equation for the n-reach set, we obtain[2]

$$X_n \subseteq \boldsymbol{A}^n X_0 \oplus U_{cb}^n. \tag{24}$$

Shifting our attention from reach sets to reach tubes, we can now over-approximate the reach tube by abstract acceleration of the summands in (24), as follows.

Theorem 3 *The abstract acceleration*

$$\hat{\tau}^{\sharp n}(X_0, U) = \mathcal{A}^n X_0 \oplus \mathcal{B}^n U_c \oplus \mathcal{B}_b^n U_b, \tag{25}$$

where $\mathcal{A}^n \supseteq \bigcup_{k\in[1,\dots,n]} \boldsymbol{A}^k$, $\mathcal{B}^n \supseteq \bigcup_{k\in[1,\dots,n]} \sum_{i=0}^{k-1} \boldsymbol{A}^i\boldsymbol{B}$, *and* $\mathcal{B}_b^n \supseteq \bigcup_{k\in[1,\dots,n]} F_b^*\left(\sum_{i=0}^{k-1} \boldsymbol{A}^i\boldsymbol{B}, \boldsymbol{x}\right)$, *denotes an over-approximation of the n-reach tube, namely* $\hat{X}_n \subseteq \hat{\tau}^{\sharp n}(X_0, U)$.

Proof From Equation (24) we have that $X_n \subseteq \boldsymbol{A}^n X_0 \oplus U_{cb}^n = \boldsymbol{A}^n X_0 \oplus U_c^n \oplus U_b^n$. Furthermore, from Equation (24) we also have $\hat{\tau}^{\sharp n}(X_0, U_c) \supseteq \bigcup_{k\in[1,\dots,n]} \boldsymbol{A}^k X_0 \oplus U_c^k$. Finally, from the definition of $\mathcal{B}_b^n$ we have $\mathcal{B}_b^n U_b \supseteq \bigcup_{k\in[1,\dots,n]} U_b^k$, hence $\hat{\tau}^{\sharp n}(X_0, U) \supseteq \hat{X}_n$. □

6.3 Combining abstract matrices

Calculating the reach set from the set of initial states and that originating from the input set separately, and then adding them together, can result in coarse over-approximations. To reduce this, we explain next how to apply abstract acceleration to the combined input-and-state spaces.

One important property of the abstract matrices $\mathcal{A}^n$, $\mathcal{B}^n$ and $\mathcal{B}_b^n$ is that they are related. In the case of parametric inputs, this correlation is linear and described by the acceleration defined in Lemma (2). In the case

[2] Note that although we are working in the eigenspace, these sets can be traced back to corresponding sets in the original state space such that $U'_b = \boldsymbol{S}^{-1}BU_b$, $U'_c = \boldsymbol{S}^{-1}BU_c$, and $U'_d = \boldsymbol{S}^{-1}BU_d$. Hence, this inclusion is also valid in the original state space.

of $\mathcal{B}_b^n$ this relationship is not linear (see Eq. 21). However, we can still find a linear over-approximation of the relation between $\mathcal{B}_b^n$ and $\mathcal{A}^n$ based on the time steps k.

Given two sets $X \in \mathbb{R}^p$ and $U \in \mathbb{R}^q$ and a transition equation $X_{k+1} = \boldsymbol{A}X_k + \boldsymbol{B}U$, which is related to $\rho_{X_{k+1}}(\boldsymbol{v}) = \rho_{AX_k}(\boldsymbol{v}) + \rho_{\boldsymbol{B}U}(\boldsymbol{v})$, we define a set

$$X' = \left\{ \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{Bu} \end{bmatrix} : \boldsymbol{x} \in X, \boldsymbol{u} \in U \right\}$$

so that

$$\rho_{X_{k+1}}(\boldsymbol{v}) = \rho_{X'_k} \begin{bmatrix} \boldsymbol{A}^\intercal \boldsymbol{v} \\ \boldsymbol{v} \end{bmatrix} = \rho_{X'_k} \left(\boldsymbol{D}^\intercal \boldsymbol{v}' \right), \text{ with } \boldsymbol{D} = \begin{bmatrix} \boldsymbol{A} & 0 \\ 0 & \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{v}' = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{v} \end{bmatrix}.$$

Accelerating X_{k+1}, we obtain

$$\rho_{X_n}(\boldsymbol{v}) = \rho_{\boldsymbol{A}^n X_0}(\boldsymbol{v}) + \rho_{(\boldsymbol{I}-\boldsymbol{A}^n)(\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}U}(\boldsymbol{v}) = \rho_{X'_0}\left((\boldsymbol{D}^n)^T \boldsymbol{v}' \right), \text{ with } \boldsymbol{D}^n = \begin{bmatrix} \boldsymbol{A}^n & 0 \\ 0 & (\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1} \end{bmatrix},$$

in the case of parametric inputs. More generally, the diagonal elements of $\boldsymbol{D}^n$ correspond to the diagonal elements of $\boldsymbol{A}^n$ and $\sum_{k=0}^{n-1} \boldsymbol{A}^k \boldsymbol{B}$, which means we can construct

$$\mathcal{D}^n = \begin{bmatrix} \mathcal{A}^n & 0 \\ 0 & \mathcal{B}^n \end{bmatrix}, \text{ so that } \rho_{X_n}(\boldsymbol{v}) = \rho_{X'_0}\left((\mathcal{D}^n)^\intercal \boldsymbol{v}' \right), \tag{26}$$

where $\mathcal{A}^n$ and $\mathcal{B}^n$ are the abstract matrices in Equations (13) and (19). We can then apply this abstraction to (21) and obtain:

$$\begin{aligned} \rho_{X_n}(\boldsymbol{v}) &= \rho_{X'_0}(\mathcal{D}_b^{nT} \boldsymbol{v}'), \text{ where} \\ \mathcal{D}_b^n &= \begin{bmatrix} \mathcal{A}^n & 0 \\ 0 & \mathcal{B}_b^n \end{bmatrix}, \quad \boldsymbol{v}' = \begin{bmatrix} \boldsymbol{v} \\ f_b(\boldsymbol{v}) \end{bmatrix}, \\ \mathcal{B}_b^n &= \boldsymbol{S}F_b^{-1}\left((\boldsymbol{I} - \mathcal{J}_b^n)(\boldsymbol{I} - \boldsymbol{J}_b)^{-1} F_b(\boldsymbol{S}^{-1}) \right), \end{aligned} \tag{27}$$

with $\boldsymbol{J}_b$ defined in (22). This model provides a tighter over-approximation than (25), since the accelerated dynamics of the inputs are now coupled to the acceleration of the dynamical part of the model.

Example 1 In order to illustrate this, let us consider the one-dimensional model $\boldsymbol{x}_{k+1} = 0.5\boldsymbol{x} + 1, \boldsymbol{x}_0 = 1$. If we calculate $\mathcal{A}$ and $\mathcal{B}$ separately we get $\hat{\boldsymbol{x}} = \bigcup_{k=0}^{\infty} \boldsymbol{A}^k \boldsymbol{x}_0 + \bigcup_{k=0}^{\infty} (1 - \boldsymbol{A}^k) \frac{\boldsymbol{u}}{1-\boldsymbol{A}} = [1, 3]$, however, using $\mathcal{D}$ we have $\hat{\boldsymbol{x}} = \bigcup_{k=0}^{\infty} \boldsymbol{A}^k \left(\boldsymbol{x}_0 - \frac{\boldsymbol{u}}{1-\boldsymbol{A}} \right) + \frac{\boldsymbol{u}}{1-\boldsymbol{A}} = [1, 2]$. □

7 Abstract Acceleration with Guards: Estimation of the Number of Iterations

In the presence of spatial guards G, we are interested in estimating the number of iterations used to calculate the abstract matrices. Since we are dealing with reach sets, we differentiate between sets that are entirely inside the guard, sets that are crossing it, and sets that are entirely outside of it. The latter reach sets should never be propagated, whereas reach sets crossing guards should be made as tight as possible.

Given a convex polyhedral guard expressed as the assertion $G = \{\boldsymbol{x} : \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\}$, we define $G_{i,*}$ as the i^{th} row of $\boldsymbol{G}$ and h_i as the corresponding element of $\boldsymbol{h}$. We denote the normal vector to the i^{th} face of the guard as $\boldsymbol{g}_i = G_i^\intercal$. The distance of the hyperplane defined by the i-th guard to the origin is thus $\gamma_i = \frac{h_i}{|\boldsymbol{g}_i|}$.

Given a convex set X, we may now describe its position with respect to each face of the guard through the use of its support function alongside the normal vector to the hyperplane (for clarity, we assume the origin to be inside set X):

$$\begin{aligned} \rho_X(\boldsymbol{g}_i) &\leq \gamma_i, && \text{inside the hyperplane,} \\ -\rho_X(-\boldsymbol{g}_i) &\geq \gamma_i, && \text{outside the hyperplane.} \end{aligned}$$

Applying this to Equation (24) we obtain:

$$\rho_{X_n}(\boldsymbol{g}_i) = \rho_{X_0}(\boldsymbol{A}^{\underline{n_i}\intercal}\boldsymbol{g}_i) + \rho_{U^n_{cb}}(\boldsymbol{g}_i) \leq \gamma_i, \tag{28}$$

$$\rho_{X_n}(-\boldsymbol{g}_i) = \rho_{X_0}(-\boldsymbol{A}^{\overline{n_i}\intercal}\boldsymbol{g}_i) + \rho_{U^n_{cb}}(-\boldsymbol{g}_i) \leq -\gamma_i. \tag{29}$$

From the inequalities above we can determine up to which number of iterations $\underline{n_i}$ the reach tube remains inside the corresponding hyperplane, and starting from which iteration $\overline{n_i}$ the corresponding reach set goes beyond the guard.

In order for a reach set to be inside the guard it must therefore be inside all of its faces, and we can ensure it is fully outside of the guard set when it is fully beyond any of them. Thus, we have $\underline{n} = \min\{\ \underline{n_i}\ \}$, and $\overline{n} = \min\{\ \overline{n_i}\ \}$.

We now discuss why these two cases are important. Looking at the transition in Equation (1), we can easily derive that if $\boldsymbol{G}\boldsymbol{x}_k \not\leq \boldsymbol{h}$ (i.e. the point lies outside at least one of the faces of the guard set), the post-image of all subsequent iterations of that point must not be included. As such, any over-approximation of the reach set will only add imprecision. Therefore, we will use the bounds $\underline{n}$ and $\overline{n}$ to create a tighter over-approximation. Let

$$\begin{aligned}
\hat{X}^\sharp_{\underline{n}} &= \mathcal{A}^{\underline{n}}X_0 \oplus \mathcal{B}_{\underline{n}}U \quad \text{(n-reach tube)}\\
X^\sharp_{\underline{n}} &= \boldsymbol{A}^{\underline{n}}X_0 \oplus \mathcal{B}_{\underline{n}}U \quad \text{(n-reach set)}\\
\hat{X}^\sharp_{\overline{n}\mid\underline{n}} &= \tau\left(\mathcal{A}^{\overline{n}-\underline{n}-1}X^\sharp_{\underline{n}} \oplus \mathcal{B}_{\underline{n}}U \cap G, U\right)\\
\hat{X}^\sharp_{\overline{n}} &= \hat{X}^\sharp_{\overline{n}\mid\underline{n}} \cup \hat{X}^\sharp_{\underline{n}}.
\end{aligned}$$

This double step prevents the set $\left\{\boldsymbol{x} : \boldsymbol{x} \in \hat{X}^\sharp_{\underline{n}}, \boldsymbol{x} \notin X^\sharp_{\underline{n}}\right\}$ to be included in further computations, thus reducing the size of the over-approximation.

Computing the maximum $\underline{n_i}$ such that Equation (28) is satisfied is not trivial, because the unknown $\underline{n_i}$ occurs in the exponent of the equation. However, since an intersection with the guard set will always return a sound over-approximation, we do not need a precise value: we can over-approximate it by decomposing $\boldsymbol{g}_i$ into the generalised eigenspace of $\boldsymbol{A}$. More precisely, let

$$\boldsymbol{g}_i = \sum_{j=1}^{p} k_{ij}\boldsymbol{v}_j + res(\boldsymbol{g}_i), \tag{30}$$

where $\boldsymbol{v}_j$ are row vectors of $\boldsymbol{S}^{-1}$ or $-\boldsymbol{S}^{-1}$ such that $k_{ij} \geq 0$, and $res(\boldsymbol{g}_i)$ is the component of the vector $\boldsymbol{g}_i$ that lies outside the range of $\boldsymbol{S}$, namely the subspace spanned by its columns. Notice that since by definition $\boldsymbol{S}$ always has an inverse, it is full rank and therefore $res(\boldsymbol{g}_i) = \boldsymbol{0}$ and subsequently not relevant. It is also important to note that $\boldsymbol{S}$ is the matrix of generalised eigenvectors of $\boldsymbol{A}$ and therefore we are expressing the guard in the generalised eigenspace of $\boldsymbol{A}$. Thus we obtain:

$$\rho_{X_0}(\boldsymbol{A}^{n\intercal}\boldsymbol{g}_i) = \rho_{X_0}\left(\sum_{j=1}^{p} k_{ij}\boldsymbol{A}^{n\intercal}\boldsymbol{v}_j\right) \leq \sum_{j=1}^{p} k_{ij}\rho_{X_0}\left(\boldsymbol{A}^{n\intercal}\boldsymbol{v}_j\right).$$

7.1 Overestimating the number of iterations of a model without inputs

Since rotating dynamics and Jordan shapes will have a complex effect on the behaviour of the model, we seek to transform the Jordan form into a real positive matrix by using the absolute value of the eigenvalues. In such a case, the support function in each direction is monotonically increasing (or decreasing), and it is therefore very easy to find a bound for its progression. We note that the envelope (described by the absolute value) of rotating dynamics will always contain the true dynamics and is therefore a sound over-approximation. We will initially assume that γ_i is positive and then extend to the general case.

Let $\rho_{X_0}(\boldsymbol{A}^{n\intercal}\boldsymbol{g}_i) = \rho_{X_0'}(\boldsymbol{J}^{n\intercal}\boldsymbol{g}_i')$, so that $\boldsymbol{g}_i' = \boldsymbol{S}^{-1}\boldsymbol{g}_i$ and

$$X_0 = \{\boldsymbol{x} \mid \boldsymbol{C}_{X_0}\boldsymbol{x} \leq \boldsymbol{d}_{X_0}\}, \quad X_0' = \boldsymbol{S}^{-1}X_0 = \{\boldsymbol{x} \mid \boldsymbol{C}_{X_0}\boldsymbol{S}\boldsymbol{x} \leq \boldsymbol{d}_{X_0}\}.$$

Further, let $\Lambda_\sigma = \{\lambda_i : i \in [1, \dots, p],\ \bigwedge_{j=1}^{i-1}(\lambda_i^* \neq \lambda_j \wedge \lambda_i \neq \lambda_j)\}$, be the set of eigenvalues with distinct values (excluding conjugate pairs and geometric multiplicities). Introduce $f_\sigma(\boldsymbol{v}) : \mathbb{R}^p \to \mathbb{R}^{p_b}$, where p_b is the cardinality of Λ_σ, such that $f_\sigma(\boldsymbol{v}) = \text{red}(\boldsymbol{v}_\sigma)$, and

$$(\boldsymbol{v}_\sigma)_i = \begin{cases} 0 & \lambda_i \notin \Lambda_\sigma \\ \sqrt{\sum_{j \in [1,\dots,p] \wedge (\lambda_j = \lambda_i \vee \lambda_j = \lambda_i^*)} \boldsymbol{v}_j^2} & \lambda_i \in \Lambda_\sigma \end{cases},$$

and furthermore let $F_\sigma : \mathbb{R}^{r\times p} \to \mathbb{R}^{r\times p_b}$ be

$$F_\sigma(\boldsymbol{C}) = \boldsymbol{C}_\sigma, \text{ where } (\boldsymbol{C}_\sigma)_{i,*} = f_\sigma(\boldsymbol{C}_{i,*}).$$

Above, red$(\cdot)$ is a function that reduces the dimension p of a vector to p_b by removing the elements $\lambda_i \notin \Lambda_\sigma$. This reduction is not strictly necessary, but it enables a faster implementation. Correspondingly, given $\boldsymbol{J} = diag\,(\boldsymbol{J}_s, s \in [1, \dots, p_b])$, we have

$$\boldsymbol{J}_\sigma = \begin{bmatrix} \overline{\sigma}_1 & 0 & \cdots & 0 \\ 0 & \overline{\sigma}_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \overline{\sigma}_r \end{bmatrix}, \tag{31}$$

where $\overline{\sigma}_s = \sup_{\boldsymbol{x}\neq 0} \frac{\|\boldsymbol{J}_s\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2}$ is the maximum singular value [48] of the Jordan block $\boldsymbol{J}_s$. Finally, let

$$\begin{aligned} \boldsymbol{x}_c' &= \frac{1}{2}(\rho_{X_0'}(\boldsymbol{v_i}) + \rho_{X_0'}(-\boldsymbol{v_i})), \quad \boldsymbol{v}_{ij} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}, \\ X_\sigma' &= \{\boldsymbol{x} \mid F_\sigma(\boldsymbol{C}_{X_0}\boldsymbol{S})f_\sigma(\boldsymbol{x}) \leq f_\sigma(\boldsymbol{d}_{X_0} - \boldsymbol{S}\boldsymbol{C}_{X_0}\boldsymbol{x}_c')\}, \\ X_0' &\subseteq f_\sigma^{-1}(X_{c\sigma}'), \text{ where } X_{c\sigma}' = \{f_\sigma(\boldsymbol{x}_c')\} \oplus X_\sigma' \end{aligned} \tag{32}$$

and $\boldsymbol{v}_\sigma = f_\sigma(\boldsymbol{v})$, where $\boldsymbol{x}_c'$ is the Cartesian centre of X_0' and $X_{c\sigma}'$ an over-approximation of X_0' centered at $\boldsymbol{x}_c'$.

Using properties of eigenvalues and of singular values, we obtain $\rho_{X_0}((\boldsymbol{A}^n)^\intercal \boldsymbol{v}_j) \leq \overline{\sigma}_j^{\,n}\rho_{X_{c\sigma}}((\boldsymbol{v}_\sigma)_j)$, where $j \in [1, \dots, p_b]$, and therefore

$$\rho_{X_0}((\boldsymbol{A}^n)^\intercal \boldsymbol{g}_i) \leq \sum_{j=1}^{p_b} k_{ij}\overline{\sigma}_j^n \rho_{X_{c\sigma}}((\boldsymbol{v}_\sigma)_j), \tag{33}$$

where k_{ij} are the coefficients in Equation (30).

Since we have assumed to have no inputs, $\rho_{U_c^n}(\boldsymbol{g}_i) + \rho_{U_b^n}(\boldsymbol{g}_i) = 0$, hence we may solve for $\underline{n}_i$ as:

$$\rho_{X_0}((\boldsymbol{A}^{\underline{n}_i})^\intercal \boldsymbol{g}_i) \leq \sum_{j=1}^{p_b} k_{ij}\overline{\sigma}_j^{\,\underline{n}_i} \rho_{X_{c\sigma}}((\boldsymbol{v}_\sigma)_j) \leq \gamma_i. \tag{34}$$

In order to separate the divergent parts of the dynamics from the convergent one, let us define

$$\overline{k}_{ij} = \max\{k_{ij}\,\rho_{X_{c\sigma}}((\boldsymbol{v}_\sigma)_j), 0\}, \quad \overline{\sigma} = \max\{\overline{\sigma}_s : s \in [1, \dots, p_b]\}.$$

This step will allow us to track effectively which trajectories are likely to hit the guard and when, since it is only the divergent element of the dynamics that can increase the size of the reach tube in a given direction. This condition requires that the set of initial conditions is also inside the guard, which is a reasonable assumption.

Replacing in Equation (34), we obtain

$$\overline{\sigma}^{\,n} \sum_{j=1}^{p} \overline{k}_{ij} \left(\frac{\overline{\sigma}_j}{\overline{\sigma}}\right)^n \leq \gamma_i\,, \tag{35}$$

which allows to finally formulate the following iteration scheme for under-approximating n.

Proposition 1 *An iterative under-approximation of the number of iterations n can be computed by starting with* $\underline{n_i} = 1$, *and iterating over*

$$\underline{n_i} \geq n = \log_{\overline{\sigma}}(\gamma_i) - \log_{\overline{\sigma}}\left(\sum_{j=1}^{p_b} \overline{k}_{ij}\left(\frac{\overline{\sigma}_j}{\overline{\sigma}}\right)^{\underline{n_i}}\right), \tag{36}$$

substituting $n_i = n$ *on the right-hand side until we meet the inequality. (If* $n < 0$ *is obtained at the first iteration, then* $\underline{n_i} = 0$ *is the obtained solution.)*

Proof Notice that the sequence $\underline{n_i}$ is monotonically increasing, before it breaks the inequality. As such any local minimum represents a sound under-approximation of the number of loop iterations. Note that in the case where $\gamma_i \leq 0$ we must first translate the system coordinates such that $\gamma_i > 0$. This is simply done by replacing $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{c}$ and operating over the resulting system where $\gamma_i' = \rho_{\boldsymbol{c}}(\boldsymbol{g}_i) + \gamma_i$.

Mathematically this is achieved as follows: first we get $\boldsymbol{c}$ by finding the centre of the interval hull (see Equation (20)) of G (if G is open in a given direction we may pick any number in that direction for the corresponding row of c). Next we transform the dynamics into

$$\begin{bmatrix}\boldsymbol{x}_k\\ \boldsymbol{1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{Ac}\\ \boldsymbol{0} & \boldsymbol{1}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_{k-1}\\ \boldsymbol{1}\end{bmatrix} + \begin{bmatrix}\boldsymbol{B}\\ \boldsymbol{0}\end{bmatrix}\boldsymbol{u}_k, \text{ where } \begin{bmatrix}\boldsymbol{x}_{k-1}\\ \boldsymbol{1}\end{bmatrix} \in \left\{\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{1}\end{bmatrix} : \begin{bmatrix}\boldsymbol{G} & \boldsymbol{Gc}\\ \boldsymbol{0} & \boldsymbol{1}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_{k-1}\\ \boldsymbol{1}\end{bmatrix} \leq \begin{bmatrix}\boldsymbol{h}\\ \boldsymbol{1}\end{bmatrix}\right\}. \quad \square$$

7.2 Underestimating the number of iterations of a model without inputs

In order to apply a similar technique to (29), we must find an equivalent under-approximation. In the case of Equation (34), the quantities $\overline{\sigma}_j$ ensure that the equation diverges faster than the real dynamics, hence the estimation found is an upper bound to the desired iteration. In this case we want the opposite, hence we look for a model where the dynamics diverge slower. It is easy to show that $\lambda_{bj} = |\lambda_j|$ represents these slower dynamics,

$$\rho_{X_0}(-\boldsymbol{A}^{\overline{n_i}\intercal}\boldsymbol{g}_i) \leq \sum_{j=1}^{p} k_{ij}\lambda_{bj}^{\overline{n_i}}\rho_{X_{c\sigma}}(-(\boldsymbol{v}_\sigma)_j) \leq -\gamma_i,$$

which reduces to

$$\overline{\sigma}^{\,n}\sum_{j=1}^{p}\underline{k}_{ij}^{-}\left(\frac{\lambda_{bj}}{\overline{\sigma}}\right)^n + \overline{\sigma}^{\,n}\sum_{j=1}^{p}\underline{k}_{ij}^{+} \leq -\gamma_i\,, \tag{37}$$

where $\underline{k}_{ij}^{-} = \min\left\{k_{ij}\,\rho_{X_{c\sigma}}(-(\boldsymbol{v}_\sigma)_j)\,,\ 0\right\}$ and $\underline{k}_{ij}^{+} = \max\left\{k_{ij}\,\rho_{X_{c\sigma}}(-(\boldsymbol{v}_\sigma)_j)\,,\ 0\right\}$.

An additional consideration must be made regarding rotational dynamics. In the previous case we did not care about the rotational alignment of the set X_n with respect to the vector $\boldsymbol{g}_i$, because any rotation would remain inside the envelope corresponding to the absolute value ($r^k cos(k\theta) \leq r^k$). In the case of an under-approximation, although the magnitude of a complex eigenvalue at a given iteration may be greater than the support of the guard under verification, its angle with respect to the normal to the support vector may cause the corresponding point to remain inside the guard. We must therefore find iterations that are aligned with the normal to the guard, thus ensuring that the chosen point is outside it. In order to do this, let us first fix the magnitudes of the powered eigenvalues, in the case of convergent dynamics we will assume they have converged a full rotation to make our equation strictly divergent. Let $\underline{\theta} = \min\{\theta_j, j \in [1, \ldots, p]\}$, where θ_j are the angles of the complex conjugate eigenvalues. Let $n_\theta = \frac{2\pi}{\underline{\theta}}$ be the maximum number of iterations needed for any of the dynamics to complete a full turn. Then at any given turn $|\lambda_j|^{\overline{n}_i+n_\theta} \leq |\lambda_j|^{\overline{n}_i+n}$, where $|\lambda_i| \leq 1$ and $n \in [0, n_\theta]$. This means that any bound we find on the iterations will be necessarily smaller than the true value. Our problem becomes the solution to:

$$\max_n\left(\overline{\sigma}^{\,\overline{n}_i}\sum_{j=1}^{p} c_{ij}\cos((n-\overline{n}_i)\theta_j - \alpha_{ij})\right), \quad \alpha_{ij} = \cos^{-1}(\boldsymbol{g}_i \cdot \boldsymbol{v}_j), \quad c_{ij} = \begin{cases}\underline{k}_{ij}^{-}\left(\frac{\lambda_{bj}}{\overline{\sigma}}\right)^{\overline{n}_i} & |\lambda_j| \geq 1\\ \underline{k}_{ij}^{-}\left(\frac{\lambda_{bj}}{\overline{\sigma}}\right)^{\overline{n}_i+n_\theta} & |\lambda_j| < 1\end{cases}.$$

The problem is simplified by soundly under-approximating the cosines and removing the constants, namely deriving successively the expressions

$$\max_n \left(\overline{\sigma}^{\,\overline{n}_i} \sum_{j=1}^{p} c_{ij} \left(1 - \frac{((n-\overline{n}_i)\theta_j - \alpha_{ij})^2}{2} \right) \right),$$
$$\min_n \left(\sum_{j=1}^{p} c_{ij}((n-\overline{n}_i)\theta_j - \alpha_{ij})^2 \right), \text{ and}$$
$$\min_n \left(\sum_{j=1}^{p} c_{ij}\theta_j^2(n-\overline{n}_i)^2 + c_{ij}\alpha_{ij}\theta_j(n-\overline{n}_i) \right).$$

The solution to the last equation is

$$n = \overline{n}_i - \frac{\sum_{j=1}^{p} c_{ij}\alpha_{ij}\theta_j}{2\sum_{j=1}^{p} c_{ij}\theta_j^2}, \text{ with } n \in [\overline{n}_i, \overline{n}_i + n_\theta]. \tag{38}$$

The second part of the equation is expected to be a positive value. When this is not the case, the dominating dynamics will have a rotation $\theta_j \geq \frac{\pi}{2}$. In such cases, we must explicitly evaluate the set up to $\left(\frac{2\pi}{\theta_j} + 1\right)$ iterations after $\overline{n}_i$, in order to ensure that we have covered a full rotation. If the resulting bound does not satisfy the original inequality: $\rho_{X_0}\left((\boldsymbol{A}^{\overline{n}_i})^\intercal \boldsymbol{g}_i\right) \geq \gamma_i$, we replace $\overline{n}_i = n$ until it does [3].

Proposition 2 *An iterative under-approximation of the number of iterations n can be computed by starting with* $\overline{n}_i' = 0$ *and iterating over*

$$\overline{n}_i' \leq n = \log_{\overline{\sigma}}(\gamma_i) - \log_{\overline{\sigma}}\left(\sum_{j=1}^{p} \underline{k}_{ij}^{-} \left(\frac{\lambda_{bj}}{\overline{\sigma}} \right)^{\overline{n}_i'} + \sum_{j=1}^{p} \underline{k}_{ij}^{+} \right),$$
$$\overline{n}_i = \overline{n}_i' + k, \textit{ given } \rho_{X_0}\left((\boldsymbol{A}^{(\overline{n}_i'+k)})^\intercal \boldsymbol{g}_i\right) \geq \gamma_i, \tag{39}$$

where k is the result of Equation (38)*. We substitute for* $\overline{n}_i = n$ *on the right-hand side as long as the first inequality holds, and then find k such that the second inequality holds.*

Since we are explicitly verifying the inequality, there is no further proof required.

7.3 Estimating the number of iterations of a model with inputs

For an LTI model with inputs, we will use the same paradigm explained in the previous section after transforming the system with inputs into an over-approximating system without inputs.

Let $X'_{c\sigma}, U'_{c\sigma}$ be the corresponding sets of initial states and inputs obtained by applying Equation (32) to X'_0 and U'_J, and let $U'_{J\sigma} = (\boldsymbol{I} - \boldsymbol{J}_\sigma)^{-1} U'_{c\sigma}$. The accelerated resulting system may be represented by the equations

$$(X'_{c\sigma})_n = \boldsymbol{J}_\sigma^n X'_{c\sigma} \oplus (\boldsymbol{I} - \boldsymbol{J}_\sigma^n) U_{J\sigma'},$$
$$\rho_{(X'_{c\sigma})_n}(\boldsymbol{v}) = \rho_{X'_{c\sigma}}\left(\boldsymbol{J}_\sigma^{nT}\boldsymbol{v}\right) + \rho_{U'_{J\sigma}}(\boldsymbol{v}) - \rho_{U'_{J\sigma}}\left(\boldsymbol{J}_\sigma^{nT}\boldsymbol{v}\right). \tag{40}$$

Let us now define $(XU)_\sigma = \{\boldsymbol{x} - \boldsymbol{u} \mid \boldsymbol{x} \in X'_{c\sigma}, \boldsymbol{u} \in U'_{J\sigma}\}$, which allows us to translate the system into

$$\rho_{((XU)'_\sigma)_n}(\boldsymbol{v}) = \rho_{(XU)'_\sigma}\left(\boldsymbol{J}_\sigma^{nT}\boldsymbol{v}\right), \tag{41}$$

which has the same shape as the equations in the previous section. We may now apply the techniques described above to find the bounds on the iterations.

[3] This is a tighter value than that in [14], where we over-approximated using $n_\theta = \frac{(2\pi)^m}{\prod_j \theta_j}$, where m is the number of conjugate pairs.

7.4 Narrowing the estimation of the number of iterations

The estimations above can be conservative, but we may obtain tighter bounds on the number of iterations. In the first instance, note that we have eliminated all negative terms in the sums in Equation (36). Reinstating these terms can result in loss of monotonicity, but we may still create an iterative approach by fixing the negative value at intermediate stages. Let $\underline{n_i}$ be our existing bound for the time horizon before reaching a guard, and $\underline{k}_{\underline{n_i}} = \sum_{j=1}^{p} \underline{k}_{ij} \left(\frac{\overline{\sigma}_j}{\overline{\sigma}}\right)^{\underline{n_i}}$, $\overline{k}_{\underline{n_i}} = \sum_{j=1}^{p} \overline{k}_{ij} \left(\frac{\overline{\sigma}_j}{\overline{\sigma}}\right)^{\underline{n_i}}$ the corresponding negative and positive terms of the equation. We may now find upper and lower bounds for $\underline{n_i}$ by replacing them in Equation (39) as:

$$\underline{n_i} \geq n_k = \log_{\overline{\sigma}}(\gamma_i) - \log_{\overline{\sigma}}\left(\overline{k}_{\underline{n_i}} + \underline{k}_{\underline{n_k}}\right), \tag{42}$$

where $\underline{n_k}$ is the bound found in the previous stage. Some steps of this process will provide an unsound result, however, every second step will provide a monotonically increasing sound bound which will be tighter than the one in Equation (36). Since the elements of the sums are convergent, we have that $n_i \geq n_k$ implies $\underline{k}_{n_i} \geq \underline{k}_{n_k}$ $\left(\text{i.e. } |\underline{k}_{n_i}| \leq |\underline{k}_{n_k}|\right)$ thus

$$\log_{\overline{\sigma}}\left(\overline{k}_{\underline{n_i}} + \underline{k}_{\underline{n_k}}\right) \geq \log_{\overline{\sigma}}\left(\overline{k}_{\underline{n_i}} + \underline{k}_{\underline{n_i}}\right),$$

which means that n_k in Equation (42) is smaller than our n in Equation (36) ($n_k \leq n \leq \underline{n_i}$ and $\underline{n_i} \geq \underline{n_k}$).

In the case of Equation (39), the explicit evaluation of the guard at each cycle executes the behaviour described here.

7.4.1 Maintaining geometric multiplicity

A second step in optimising the number of iterations comes from adding granularity to the bounding abstraction by retaining the geometric multiplicity using the matrix $\boldsymbol{J}_b$ (see Equation (22)).

Lemma 3 *Given a matrix $\boldsymbol{A}$ with eigenvalues $\{\lambda_s, s \in [1, \dots, r]\}$, where each eigenvalue λ_s has a geometric multiplicity p_s and corresponding generalised eigenvectors $\{\boldsymbol{v}_{s,i}, i \in [1, \dots, p_s]\}$,*

$$\forall n \geq 0, \boldsymbol{A}^n \boldsymbol{v}_s^i = \lambda_s^n \boldsymbol{v}_{s,i} + \sum_{j=1}^{i-1} \binom{n}{j} \lambda_s^{n-j} \boldsymbol{v}_{s,i-j} = \lambda_s^n \left(\boldsymbol{v}_{s,i} + \sum_{j=1}^{i-1} \binom{n}{j} \lambda_s^{-j} \boldsymbol{v}_{s,i-j} \right). \tag{43}$$

Proof By definition, given an eigenvector $\boldsymbol{v}_s$ of $\boldsymbol{A}$, then $\boldsymbol{A}\boldsymbol{v}_s = \lambda_s \boldsymbol{v}_s$ [42]. Similarly, a generalised eigenvector $\boldsymbol{v}_{s,i}$ of $\boldsymbol{A}$ satisfies the equation $(\boldsymbol{A} - \lambda_s \boldsymbol{I})\, \boldsymbol{v}_{s,i} = \boldsymbol{v}_{s,i-1}$[4]. and $\boldsymbol{v}_{s,1} = \boldsymbol{v}_s$ hence

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{v}_{s,i} &= \lambda_s \boldsymbol{v}_{s,i} + \boldsymbol{v}_{s,i-1} \\
\boldsymbol{A}^n \boldsymbol{v}_{s,1} &= \lambda_s^n \boldsymbol{v}_{s,1} \\
\boldsymbol{A}^n \boldsymbol{v}_{s,i} &= \boldsymbol{A}^{n-1}(\lambda_s \boldsymbol{v}_{s,i} + \boldsymbol{v}_{s,i-1}) = \lambda_s \boldsymbol{A}^{n-1} \boldsymbol{v}_{s,i} + \boldsymbol{A}^{n-1} \boldsymbol{v}_{s,i-1} \\
&= \lambda_s^2 \boldsymbol{A}^{n-2} \boldsymbol{v}_{s,i} + \lambda_s \boldsymbol{A}^{n-2} \boldsymbol{v}_{s,i-1} + \boldsymbol{A}^{n-1} \boldsymbol{v}_{s,i-1} = \cdots = \lambda_s^n \boldsymbol{v}_{s,i} + \sum_{j=0}^{n-1} \lambda_s^j \boldsymbol{A}^{n-j-1} \boldsymbol{v}_{s,i-1}.
\end{aligned}$$

From here we recursively expand the formula for $\boldsymbol{A}^{n-j-1} \boldsymbol{v}_{s,i-1}$ and obtain:

$$\begin{aligned}
\boldsymbol{A}^n \boldsymbol{v}_{s,i} &= \lambda_s^n \boldsymbol{v}_{s,i} + \sum_{j=0}^{n-1} \lambda_s^j \lambda_s^{n-j-1} \boldsymbol{v}_{s,i-1} + \sum_{j=0}^{n-1} \sum_{k=0}^{n-2} \lambda_s^k \boldsymbol{A}^{n-k-2} \boldsymbol{v}_{s,i-2} \\
&= \lambda_s^n \boldsymbol{v}_{s,i} + n\lambda_s^{n-1} \boldsymbol{v}_{s,i-1} + n \sum_{j=0}^{n-2} \lambda_s^j \boldsymbol{A}^{n-j-2} \boldsymbol{v}_{s,i-2} = \cdots = \lambda_s^n \boldsymbol{v}_{s,i} + \sum_{j=1}^{i-1} \binom{n}{j} \lambda_s^{n-j} \boldsymbol{v}_{s,i-j}. \quad \square
\end{aligned}$$

[4] More generally $(\boldsymbol{A} - \lambda_s \boldsymbol{I})\, \boldsymbol{v}_{s,i} = k_{s,i} \boldsymbol{v}_{s,i-1}$, since any vector $k\boldsymbol{v}_{s,i-1}$ is also a generlized eigenvector of $\boldsymbol{A}$, but we select generalized eigenvectors such that $k = 1$.

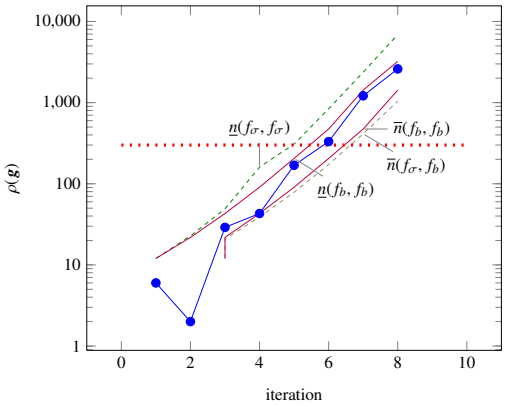

Fig. 10 Progression of the support function of a system for a given guard. The thick blue dots are real values. The dashed green line over-approximates the progression using singular values (sec 7.1), the dashed yellow line under-approximates them using eigenvalue norms (sec 7.2), whereas the continuous purple lines represent the tighter over-approximation maintaining the geometric multiplicity (sec 7.4.1). We can see how the purple line finds a better bound for $\underline{n}_i$, while the $\overline{n}_i$ bound is conservative for both approaches.

Let i' denote the position of $f_b(\lambda_j)$ within the block $\boldsymbol{J}_{bs}$ it belongs to, such that its corresponding generalised eigenvector is identified as $\boldsymbol{v}_{bs,i'} = f_b(\boldsymbol{v}_j)$. Then

$$\begin{aligned}\rho_{X_0'}(\boldsymbol{J}^{n\intercal}\boldsymbol{g}_i') &\leq \sum_{j=1}^{p_b} k_{ij}\rho_{X_0}\left(\boldsymbol{J}_b^{n\intercal} f_b(\boldsymbol{v}_j)\right) \leq \sum_{j=1}^{p_b} k_{ij}\lambda_{bj}^{n}\rho_{X_0}\left(\boldsymbol{v}_{bs,i'} + \sum_{k=1}^{i'-1}\binom{n}{k}\lambda_{bj}^{-k}\boldsymbol{v}_{bs,i'-k}\right)\\ &\leq \sum_{j=1}^{p_b} k_{ij}\lambda_{bj}^{n}\left(\rho_{X_0}\left(\boldsymbol{v}_{bs,i'}\right) + \sum_{k=1}^{i'-1}\binom{n}{k}\lambda_{bj}^{-k}\rho_{X_0}\left(\boldsymbol{v}_{bs,i'-k}\right)\right) \leq \sum_{j=1}^{p_b} k'_{ij0}\lambda_{bj}^{n} + \sum_{m=1}^{i'} k'_{ijm}\lambda_{bj}^{n}\prod_{m=0}^{p_s-i'-1}(n-m). \quad (44)\end{aligned}$$

In order to manage the product on the right hand side we use slightly different techniques for over- and under-approximations. For $\underline{n}_i$ we first find an upper bound $\underline{n}_i'$ using equation (36) and $k_{ij} = k'_{ij0} + k'_{ijm}$, and then do a second iteration using $k_{ij} = k'_{ij0} + k'_{ijm}\prod_{m=0}^{p_s-i'-1}(\underline{n}_i' - m)$, which ensures the true value is under the approximation. In the case of $\overline{n}_i$, we also start with $k_{ij} = k'_{ij0} + k'_{ijm}$ and update it during the iterative process.

Example 2 Let us look at the following example, comprising matrices:

$$\boldsymbol{J} = \begin{bmatrix} 3&0&0&0&0&0\\ 0&2&1&0&0&0\\ 0&0&2&0&0&0\\ 0&0&0&-2&0&0\\ 0&0&0&0&-1&1\\ 0&0&0&0&-1&-1 \end{bmatrix},\ \boldsymbol{S} = \begin{bmatrix} 1&0&0&0&0&0\\ 0&3&0&0&0&0\\ 0&-4&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&1&1 \end{bmatrix},\ \boldsymbol{J}_\sigma = \begin{bmatrix} 3&0&0&0\\ 0&3&0&0\\ 0&0&2&0\\ 0&0&0&\sqrt{2} \end{bmatrix},$$

with $\boldsymbol{J}_\sigma$ calculated as in Equation (31), initial condition $\boldsymbol{x}_0' = \begin{bmatrix}1&1&1&1&1&1\end{bmatrix}$ and guard set $\boldsymbol{G}\boldsymbol{x} \leq 300$ where

$$\boldsymbol{G} = \begin{bmatrix}1&3&-3&2&4&1\end{bmatrix} = \begin{bmatrix}1&1&1&2&4&-3\end{bmatrix}\boldsymbol{S}^\intercal.$$

The progression of the support function of the reach sets along this vector and the corresponding bounds, as described in the previous section, are shown in Figure 10.

Changing the eigenvalues to:

$$\boldsymbol{J} = \begin{bmatrix} 2e^{-0.2i} & 0 & 0 & 0 & 0 & 0\\ 0 & 2e^{0.2i} & 0 & 0 & 0 & 0\\ 0 & 0 & \sqrt{2}e^{-0.3i} & 0 & 0 & 0\\ 0 & 0 & 0 & \sqrt{2}e^{0.3i} & 0 & 0\\ 0 & 0 & 0 & 0 & 1.1e^{0.5i} & 0\\ 0 & 0 & 0 & 0 & 0 & 1.1e^{-0.5i} \end{bmatrix},$$

we obtain the results in Figure 11. In this second case we can see that the rotational dynamics force an increase of the initially calculated iteration to account for the effects of the rotation. □

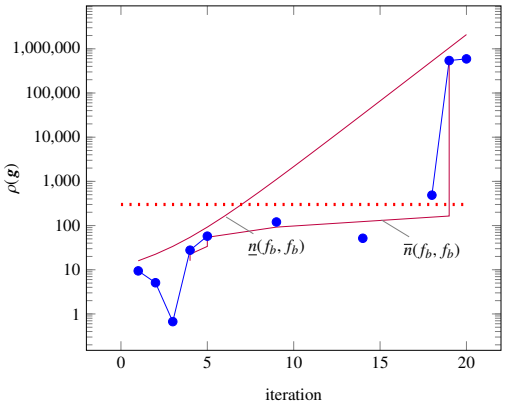

Fig. 11 Progression of the support function of a rotational system for a given guard. The thick blue dots are real values (negative values are missing due to the log scale). Continuous purple lines represent the over-approximation. The steep vertical line at iteration 19 is due to the alignment of the rotations with the guard at this point. The point at iteration 14 appears below the line because of the higher point at iteration 9. The procedure will either find that this boundary was met at iteration 9 or push it forward to iteration 19.

7.5 Case study

We have selected a known benchmark from the literature to illustrate the discussed procedure: the room temperature control problem [27]. The temperature (variable `temp`) of a room is controlled via a user-defined input (`set`), which can be changed at any discrete time step through a heating (`heat`) element, and is affected by ambient temperature (`amb`) that is out of the control of the model.

We formalise the description of such a system both via a linear loop and with a dynamical model. Observe that since such a system may be software controlled, Algorithm 1 shows a pseudo-code fragment for the temperature control problem. We use the `read` function to represent non-deterministic values between 0 and the maximum

Algorithm 1 Temperature Control Loop

States: temp=temperature, heat=heat output.
Inputs: set=set-point, amb=ambient temperature.

```
1: temp=5+read(35);
2: heat=read(1);
3: while(temp< 400 && heat< 300)
4: {
5:       amb=5+read(35);
6:       set=read(300);
7:       temp=.97 temp + .02 amb + .1 heat;
8:       heat=heat + .05 set;
9: }
```

given as its argument. Alternatively, this loop corresponds to the following hybrid dynamical model:

$$\begin{bmatrix} temp \\ heat \end{bmatrix}_{k+1} = \begin{bmatrix} 0.97 & 0.1 \\ -0.05 & 1 \end{bmatrix} \begin{bmatrix} temp \\ heat \end{bmatrix}_k + \begin{bmatrix} 0.02 & 0 \\ 0 & 0.05 \end{bmatrix} \begin{bmatrix} amb \\ set \end{bmatrix}_k,$$

with initial condition $\begin{bmatrix} temp \\ heat \end{bmatrix}_0 \in \begin{bmatrix} [5,40] \\ [0,1] \end{bmatrix}$, non-deterministic inputs $\begin{bmatrix} amb \\ set \end{bmatrix}_k \in \begin{bmatrix} [5,40] \\ [0,300] \end{bmatrix}$, and guard set $G = \left\{ \begin{bmatrix} temp \\ heat \end{bmatrix} : \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} temp \\ heat \end{bmatrix} < \begin{bmatrix} 400 \\ 300 \end{bmatrix} \right\}$.

In this model the variables are continuous and take values over the real line, whereas within the code they are represented as long double precision floating-point values, with precision of $\pm 10^{-19}$, moreover the error of the approximate Jordan form computation results in $\delta_{max} < 10^{-17}$. The eigendecomposition of the dynamics is (the values are rounded to three decimal places):

$$\begin{aligned}
\boldsymbol{A} &= \boldsymbol{SJS}^{-1} \subseteq \mathbb{SJS}^{-1}, \text{ where} \\
\mathbb{S} &= \begin{bmatrix} 0.798 \pm 10^{-14} & 0.173 \pm 10^{-15} \\ 0 \pm 10^{-19} & 0.577 \pm 10^{-14} \end{bmatrix}, \\
\mathbb{J} &= \begin{bmatrix} 0.985 \pm 10^{-16} & 0.069 \pm 10^{-17} \\ -0.069 \pm 10^{-17} & 0.985 \pm 10^{-16} \end{bmatrix}, \\
\mathbb{S}^{-1} &= \begin{bmatrix} 1.253 \pm 10^{-12} & -0.376 \pm 10^{-13} \\ 0 \pm 10^{-18} & 1.732 \pm 10^{-12} \end{bmatrix}.
\end{aligned}$$

The discussed over-approximations of the reach-sets indicate that the temperature variable intersects the guard set G at iteration $\underline{n} = 32$. Considering the pseudo-eigenvalue matrix along these iterations, we use Equation (13) to find that the corresponding complex pair remains within the following boundaries:

$$\mathcal{A}^{32} = \begin{bmatrix} r & i \\ -i & r \end{bmatrix} \begin{cases} 0.4144 < r < 0.985 \\ 0.0691 < i < 0.7651 \\ 0.1082 < r + i < 1.247 \\ 0.9159 < i - r < 0.9389 \end{cases}, \quad \mathcal{B}^{32} = \begin{bmatrix} r & i \\ -i & r \end{bmatrix} \begin{cases} 1 < r < 13.41 \\ 0 < i < 17.98 \\ 1 < r + i < 29.44 \\ 6.145 < i - r < 6.514 \end{cases}.$$

The reach tube is calculated by multiplying these abstract matrices with the initial sets of states and inputs, as described in Equation (25), by the following inequalities:

$$\hat{X}^{\#}_{32} = \mathcal{A}^{32} \begin{bmatrix} [5, 40] \\ [0, 1] \end{bmatrix} + \mathcal{B}^{32} \begin{bmatrix} [5, 40] \\ [0, 300] \end{bmatrix} = \begin{bmatrix} temp \\ heat \end{bmatrix} \begin{cases} -24.76 < temp < 394.5 \\ -30.21 < heat < 253 \\ -40.85 < temp + heat < 616.6 \\ -86.31 < temp - heat < 843.8 \end{cases}.$$

The negative values represent the lack of restriction in the code on the lower side and correspond to a cooling system (negative heating). The set is displayed in Figure 12, where for the sake of clarity only 8 directions of the 16 constraints are shown. This results in a rather tight over-approximation which, for comparison's sake, is not looser than the convex hull of all reach sets obtained by [30] using the given directions. Figure 12 further displays the initial set in black colour, the collection of reach sets in white colour, the convex hull of all reach sets in dark blue (as computed by [30]), and finally the abstractly accelerated set in light yellow (dashed lines). The outer lines represent the guards for G.

8 Application of Abstraction-Refinement to Abstract Acceleration

One of the main limitations of abstract acceleration is that, despite being very fast, it leverages an over-approximation of the actual reach tube for verification. In many cases this over-approximation can be too coarse (i.e. imprecise) for the proof of the safety property of interest. This section deals with methods for refining this over-approximation. Refinements are based on counterexamples, namely vertices of the abstract matrix that lay outside the projection of the safety specification onto the abstract space (calculated using the inverse of the reachability transformations). Our approach can be seen as an instance of the known CounterExample Guided Abstraction Refinement (CEGAR) paradigm [20].

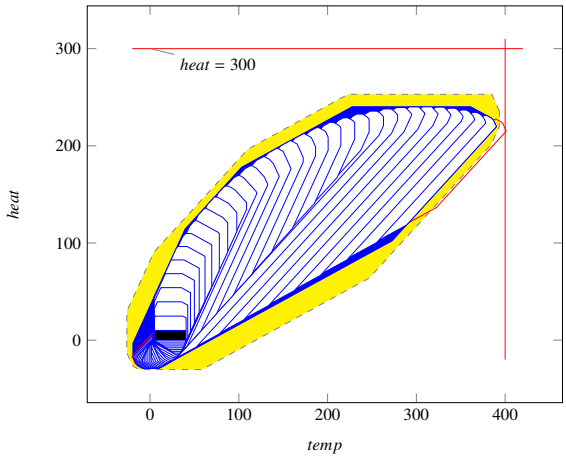

Fig. 12 The abstractly accelerated tube (yellow, dashed boundary), representing an over-approximation of the thermostat reach tube (dark blue). The set of initial conditions is shown in black, whereas successive reach sets are shown in white. The guards and the reach set that crosses them are close to the boundary in red.

8.1 Finding counterexample iterations

Because the objective is to refine the abstract dynamics, we need to find the iterations corresponding to the counterexample (i.e., the ones used to calculate the hyperplanes forming the unsafe vertex). This will allow us to find an interpolant iteration that will reduce the polyhedron in the right direction. Since the abstract dynamics are built over pairs of eigenvalues, it is possible that different eigenvalue pairs provide different results for the counterexample iteration, in which case all of them are used. Let a verification clause explore the solution $\rho_{\mathcal{A}}(\boldsymbol{v}) = s \leq \overline{s}$, where $\boldsymbol{v}$ is the direction we are examining, s its corresponding support function, and $\rho_{\mathcal{A}}(\boldsymbol{v}) \leq \overline{s}$ the safety specification. If $s > \overline{s}$ the specification will not be met and we need a refinement. Let $\boldsymbol{a}_v \in \mathcal{A}$ be the vertex at which the maximum is found, i.e., $\boldsymbol{a}_v \cdot \boldsymbol{v} = s$. The iterations corresponding to this counterexample may be found by analysing the dynamics of each pair of eigenvalues independently, and finding the point closest to the hyperplane whose inequality has been violated. This is done as follows:

1. Conjugate eigenvalues
   Since the trajectories along these are circular and centred at the origin, we can find the angle that $\boldsymbol{a}_v$ forms with the axes of the eigenvalues and use it to calculate the right iteration. Let θ_i be the angle of the conjugate eigenvalue pair and $\theta_{\boldsymbol{a}_v}(i)$ the angle formed by $\boldsymbol{a}_v$ in the i^{th} plane (this is equivalent to $\tan^{-1}\left(\frac{(\boldsymbol{a}_v)_i}{(\boldsymbol{a}_v)_{i+1}}\right)$). The corresponding iteration will depend on whether the eigenvalue is convergent or divergent. In the former case, it will be $\frac{\theta_{\boldsymbol{a}_v}(i)}{\theta_i}$, and in the latter it will be $(n - (n \mod \frac{1}{\theta_i})) + \frac{\theta_{\boldsymbol{a}_v}(i)}{\theta_i}$, where $\mod$ is the modulus operation over the reals.
2. Real eigenvalues
   In the case of reals, finding the iteration relies on the direct relation between the given eigenvalue and the target counterexample. Since $(\boldsymbol{a}_v)_i \approx \lambda_i^k \Rightarrow k \approx \log_{\lambda_i}((\boldsymbol{a}_v)_i)$. If the logarithm does not exist, then we presume we cannot further refine using this method.
3. Jordan blocks with non-unitary geometric multiplicity
   In the case of larger Jordan blocks we need to examine the nature of the dynamics. Let us look at the equation representing the contribution of a Jordan block to the support:

$$\rho_{\lambda_s}(\boldsymbol{v}) = \sum_{j=0}^{p_s} \binom{n}{j} \lambda^{n-j} \boldsymbol{v}_{s\,j}. \tag{45}$$

   In this case we must use an iterative approximation, as described in Section 7.4.1, to find the closest iteration to the unsafe guard. Although this process is more costly than the ones described above, it is also more

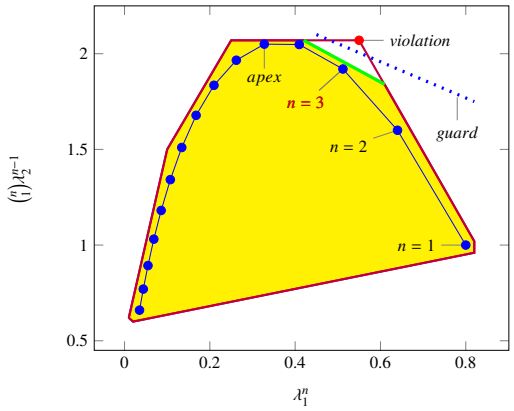

Fig. 13 Polyhedral faces from an $\mathbb{R}^2$ Jordan block subspace $(\lambda_1^n, \binom{n}{1}\lambda_2^{n-1})$ where $\lambda_1 = 0.8, \lambda_2 = 0.8$, and $1 \leq n \leq 15$. The red dot specifies an abstract vertex violating the safety specification (dashed blue line). The closest iteration to the violating vertex is $n = 3$. A new support function (green) based on $n = 3$ eliminates the violating vertex. The new abstract polyhedron meets the safety specification (yellow).

precise, thus providing a much better refinement. Note that the technique can be applied to the full set of eigenvalues or to any subset of Jordan blocks. This choice is a compromise between precision and speed. We also note that when the refinement process is done in the eigenspace, the new eigenvectors are now the identity set, which makes the problem more tractable.

Since the exclusion of an unsafe vertex from the abstract dynamics does not ensure a sufficiently tight over-approximation, we must perform this step iteratively until either we run out of new refinements or have a user-defined timeout. Once the candidate iterations are found, it suffices to add further constraints to the abstract matrix for these iterations as described in Figure 13. Notice that given the above procedure, it is often faster and more beneficial to begin by performing the refinement over the complex eigenvalues by directly examining the vector of directions $\boldsymbol{v}$ in the corresponding sub-spaces.

8.2 Using bounded concrete runs

Due to the complex interactions in the dynamics, the above procedure may not always find the correct iterations for refinement, or at least not optimal ones. For this reason, a second method is proposed, which in most cases will be more efficient and precise when the dynamics are strictly convergent. This second approach relies on the direct calculation of the initial $\bar{k}$ iterations. Since we operate over the eigenvalues and we limit $\bar{k}$ to a conservative bound, this is a relatively inexpensive calculation. The approach leverages the idea that for convergent dynamics, counterexamples are often found in the initial runs. The first step is to directly calculate the trajectory of the counterexample for the first $\bar{k}$ iterations, and its corresponding support function in the direction of $\boldsymbol{v}$. Once again, because this is a single point and a bounded time, this operation is comparatively inexpensive. The second step consists of finding an upper bound for all subsequent iterations, which we can do by using the norms of the eigenvalues and the peaks of each geometrical multiple of a Jordan Block (which relate to these norms). By selecting the larger between these two supports, we ensure soundness over the infinite time horizon. This is equivalent to evaluating the reach tube as $X_n^\sharp = \bigcup_{k=0}^{\bar{k}} \boldsymbol{A}^k X_0 \cup \mathcal{A}_{n-\bar{k}} \boldsymbol{A}^{\bar{k}} X_0$.

Since the above result is know to be an upper bound for the support in the direction of $\boldsymbol{v}$ we can directly add it to the inequalities of $\mathcal{A}$.

Refinement	Guard	Sound	Inputs	Bits	Time	Bound
No refinement	.015	No	V	80	9s	0.013693
Refinement	.005	No	V	80	18s	0.004996
No refinement	.015	No	P	128	13s	0.013633
No refinement	.015	No	V	128	24s	0.013716
Refinement	.005	No	P	128	29s	0.004996
Refinement	.005	No	V	128	48s	0.004976
Refinement	.005	No	P	1024	66s	0.004996
Refinement	.005	No	V	1024	90s	0.004999
Refinement	.005	Yes	P	1024	190s	0.004996
Refinement	.005	Yes	V	1024	563s	0.004998

Table 1 Axelerator performance on 48 dimensional Building Benchmark using various settings. P=Parametric, V=Time-varying. We see that in this case, the refinement phase doubles the analysis time. We also show the results of Counterexample Refined Abstract Acceleration for different types of inputs, bit lengths and soundness.

8.3 Case study

We have taken an industrial benchmark 'CAFF problem Instance: Building'[5]. The benchmark consists of a continuous model with 48 state variables and 1 input. Furthermore, there is an initial state corresponding to a 10-dimensional hyper-rectangle. Time is discretised using a 5 ms sample interval to give us a discrete time model for verification. Notice that the choice of sample time has very little effect on abstract acceleration. It mainly affects the requirement for floating point precision (as very small angles may require higher precision), and may have an effect on counterexample generation which can either decrease precision or increase time (i.e. we may relax the precision of the algorithm to gain speed or tightening at the cost of higher computation times). The provided model requires an analysis on the 25^{th} variable, with a safety specification requiring it to remain below .005. The problem has been verified using SpaceEx[6] for bounded time (20s) in under 3 seconds. Axelerator was run on this benchmark using different parameters. We used an Intel 2.6 GHz I7 processor with 8 GB of RAM running on Linux. Although the algorithm lends itself to concurrency, the tool currently supports only single threading. The process itself uses 82 MB on this particular benchmark. The results are summarised in Table 1. Since many tools in this area use unsound arithmetics, we present results for both sound and unsound arithmetics using abstract acceleration to show the cost of soundness. It is worth noting that for precisions under 1024 bits the tool returns soundness violation errors when using sound arithmetics.

We note in these results that the performance of Axelerator depends largely on the required level of refinement. State of the art tools can do bounded model checking faster than Axelerator, largely due to implementation optimizations (we expect a better simplex engine would allow us to be more competitive in this regard). This advantage disappears as soon as we require a larger time horizon for verification.

9 Experimental results

The overall Abstract Acceleration procedure has been implemented in C++ using the eigen-algebra package (v3.2), with multiple precision floating-point arithmetic, and has been tested on a 1.6 GHz core 2 duo computer. Unless otherwise specified, we use the sound version of Abstract Acceleration without abstraction-refinement (as per Sec. 8). The tool, called *Axelerator*, and the corresponding benchmarks used to test it (including specifications of the initial states, input ranges and guard sets), are available at

`http://www.cprover.org/LTI/`

[5] `http://cps-vo.org/node/30277`

[6] `http://spaceex.imag.fr`

	characteristics				new bounds		analysis time [sec]		
name	type	dim	inputs	bounds	IProc	Sting	IProc	Sting	mpfr
parabola_i1	$\neg s,\neg c,g$	2	1	80	+25(31%)	+28(35%)	0.007	237	0.049
parabola_i2	$\neg s,\neg c,g$	2	1	80	+24(30%)	+35(44%)	0.008	289	0.072
cubic_i1	$\neg s,\neg c,g$	3	1	120	+44(37%)	+50(42%)	0.015	704	0.097
cubic_i2	$\neg s,\neg c,g$	3	1	120	+35(30%	+55(45%)	0.018	699	0.124
oscillator_i0	$s,c,\neg g$	2	0	56	+24(43%)	+24(43%)	0.004	0.990	0.021
oscillator_i1	$s,c,\neg g$	2	0	56	+24(43%)	+24(43%)	0.004	1.060	0.024
inv_pendulum	$s,c,\neg g$	4	0	16	+8(50%)	+8(50%)	0.009	0.920	0.012
convoyCar2_i0	$s,c,\neg g$	3	2	12	+9(75%)	+9(75%)	0.007	0.160	0.043
convoyCar3_i0	$s,c,\neg g$	6	2	24	+15(62%)	+15(62%)	0.010	0.235	0.513
convoyCar3_i1	$s,c,\neg g$	6	2	24	+15(62%)	+15(62%)	0.024	0.237	0.901
convoyCar3_i2	$s,c,\neg g$	6	2	24	+15(62%)	+15(62%)	0.663	0.271	1.416
convoyCar3_i3	$s,c,\neg g$	6	2	24	+15(62%)	+15(62%)	0.122	0.283	2.103

type: s – stable loops, c – complex eigenvalues, g – loops with guard; **dim**: model dimension (variables); **bounds**: number of half-planes defining the reach tube; **new bounds**: number of bounds newly detected by Axelerator (mpfr) over the existing tools (IProc, Sting); Axelerator detects all bounds, therefore there are no lost bounds vs existing tools. **IProc** is [43]; **Sting** is [21]; **mpfr** is Axelerator (this work) using 256 bit mantissa;

Table 2 Experimental comparison of unbounded-time analysis tools with inputs

	characteristics			improved		analysis time (sec)					
name	type	dim	bounds	tighter	looser	J	(jcf)	mpfr+(jcf)		mpfr	ld
parabola_i1	$\neg s,\neg c,g$	3	80	+4(5%)	0(0%)	2.51	(2.49)	0.16	(0.06)	0.097	0.007
parabola_i2	$\neg s,\neg c,g$	3	80	+4(5%)	0(0%)	2.51	(2.49)	0.26	(0.06)	0.101	0.008
cubic_i1	$\neg s,\neg c,g$	4	120	0(0%)	0(0%)	2.47	(2.39)	0.27	(0.20)	0.110	0.013
cubic_i2	$\neg s,\neg c,g$	4	120	0(0%)	0(0%)	2.49	(2.39)	0.32	(0.20)	0.124	0.014
oscillator_i0	$s,c,\neg g$	2	56	0(0%)	-1(2%)	2.53	(2.52)	0.12	(0.06)	0.063	0.007
oscillator_i1	$s,c,\neg g$	2	56	0(0%)	-1(2%)	2.53	(2.52)	0.12	(0.06)	0.078	0.008
inv_pendulum	$s,c,\neg g$	4	12	+8(50%)	0(0%)	65.78	(65.24)	0.24	(0.13)	0.103	0.012
convoyCar2_i0	$s,c,\neg g$	5	12	+9(45%)	0(0%)	5.46	(4.69)	3.58	(0.22)	0.258	0.005
convoyCar3_i0	$s,c,\neg g$	8	24	+10(31%)	-2(6%)	24.62	(11.98)	3.11	(1.01)	0.552	0.051
convoyCar3_i1	$s,c,\neg g$	8	24	+10(31%)	-2(6%)	23.92	(11.98)	4.94	(1.01)	0.890	0.121
convoyCar3_i2	$s,c,\neg g$	8	24	+10(31%)	-2(6%)	1717.00	(11.98)	6.81	(1.01)	1.190	0.234
convoyCar3_i3	$s,c,\neg g$	8	24	+10(31%)	-2(6%)	1569.00	(11.98)	8.67	(1.01)	1.520	0.377

type: s – stable loops, c – complex eigenvalues, g – loops with guard; **dim**: model dimension; **bounds**: no. of half planes defining the reach tube; **improved**: number of bounds (and percentage) that were tighter (better) or looser (worse) than J;
J is Schrammel et al. [44]; **mpfr+** is this work using 1024bit mantissa ($e < 10^{-152}$); **mpfr** uses a 256bit mantissa ($e < 10^{-44}$); **ld** uses a 64bit mantissa ($e < 10^{-11}$); here e is the accumulated error of the dynamical system; **jcf**: time taken to compute the Jordan form

Table 3 Experimental comparison with previous work

Since many tools require the specification of a set of directions in which to perform verification, we have chosen to use an octahedral template set. That is the set of vectors that run either along an axis or at a 45 degree angle between two axes (equivalent to a set of inequalities $\pm x_i \leq bound_k$ or $\pm x_i \pm x_j < bound_k$). We also make use of interval hulls, which are defined as the smallest hyper-boxes containing a set (equivalent to $\pm x_i \leq bound_k$).

9.1 Comparison with other unbounded-time approaches

In a first experiment we have benchmarked our implementation against the tools InterProc [43] and Sting [21]. We have tested these tools on different discrete time models, including guarded/unguarded ones, stable/unstable ones, and models with complex/real loops with inputs (details in Table 2). In the first instance, we can see that Axelerator is more successful in finding tight over-approximations that remain very close to the actual reach set. InterProc and Sting are each unable to find nearly 40% of the bounds (i.e. supports), meaning they report an infinite reach-space in the corresponding directions. In these instances, InterProc (due to the limitations related

to widening) and STING (due to the absence of tight polyhedral, inductive invariants) are unable to infer finite bounds at all. This comes at a reasonable trade-off in speed performance. Axelerator is approximately 10 times slower than InterProc, and in most cases faster than Sting.

Table 3 sets up a comparison of our implementation using different levels of precision (long double, 256 bit, and 1024 bit floating-point precision) with the core procedure for abstract acceleration of linear loops without inputs (J) [44] (we use a version without inputs of the parabola, cubic and convoy models). This shows that our implementation gives tighter over-approximations on most benchmarks (column 'improved'). While on a limited number of instances the current implementation is less precise (the lower right portion of Figure 4 where the dotted red line crosses to the inside of our support gives a hint why this is happening), the overall increased precision results from mitigating the limitation on chosen directions caused by the use of logahedral abstractions (as done in previous work [44]).

At the same time, our implementation is faster than [44], partly due to the use of numeric eigendecomposition (as opposed to symbolic), but mostly in view of the cost of increasingly large rational representations in [44] needed to maintain precision when evaluating higher order systems. The fact that many bounds have improved with the new approach, while speed has increased by several orders of magnitude, provides evidence of the advantages of the new approach.

The speed-up is due to the faster Jordan form computation, which takes between 2 and 65 seconds for [44] (using the ATLAS package), whereas our implementation requires at most one second. For the last two benchmarks, the polyhedral computations blow up in [44], whereas our support function approach shows only moderately increasing runtimes. The increase of speed is owed to multiple factors, as detailed in Table 4. The difference in precision of using long double vs. multiple precision floating-point arithmetic is negligible, as all results in the given examples match exactly to 9 decimal places.

9.2 Comparison with bounded-time approaches

In a third experiment, we compare our method with the LGG algorithm [38] used by SPACEEX [30]. Since LGG provides the tightest supports for a given set of directions, this study indicates the quality of our over-approximation. In order to set up a fair comparison, we have provided the implementation of the native algorithm in [38] within our software. We have run both methods on the convoyCar example [44] with inputs, which presents an unguarded, scalable (the dimension can be increased by adding more cars), stable loop with complex dynamics, and we have focussed focused on octahedral templates. For convex reach tubes, the approximations computed by Abstract Acceleration are reasonably tight in comparison to those computed by the LGG algorithm (See Table 5). However, by storing finite disjunctions of convex polyhedra, the LGG algorithm is able to generate non-convex reach tubes, which are arguably tighter in case of oscillating or spiralling dynamics. Still, in many applications abstract acceleration can provide a tight over-approximation for the convex hull of (possibly non-convex) reach tubes.

Table 5 provides the quantitative outcomes of this comparison. For simplicity, we present only the projection of the bounds along the variables of interest on a lower dimensional model. As expected, the LGG algorithm

Optimisation	Speed-up
Eigen vs. ATLAS [7]	2–10
Support functions vs. octahedral templates	2–40
long double vs. multiple precision arithmetic	5–200
interval vs. regular arithmetic	.2–.5
Total	4–80000

Table 4 Performance improvements over [44].

	Axelerator		Axelerator using LGG		
name	100 iterations	unbounded	100 iterations	200 iterations	300 iterations
run time	166 ms	166 ms	50 ms	140 ms	195 ms
car acceleration	[-0.820, 1.31]	[-1.262, 1.31]	[-0.815, 1.31]	[-0.968, 1.31]	[-0.968, 1.31]
car speed	[-1.013, 5.11]	[-4.515, 6.15]	[-1.013, 4.97]	[-3.651, 4.97]	[-3.677, 4.97]
car position	[43.7, 83.4]	[40.86, 91.9]	[44.5, 83.4]	[44.5, 88.87]	[44.5, 88.87]

Table 5 Comparison on convoyCar2 benchmark [44], between this work and the LGG algorithm from [38]. The intervals show the minimum and maximum values obtained for each variable.

	Axelerator		Axelerator using [49]	
name	100 iterations	unbounded	100 iterations	unbounded
run time	166 ms	166 ms	155085 ms	155085 ms
car acceleration	[-0.820, 1.31]	[-1.262, 1.31]	[-24.24, 23.9]	[-∞ ∞]
car speed	[-1.013, 5.11]	[-4.515, 6.15]	[-97.2, 86.7]	[-∞ ∞]
car position	[43.7, 83.4]	[40.86, 91.9]	[-319, 343.4]	[-∞ ∞]

Table 6 Comparison on convoyCar2 benchmark [44], between this work and the work in [49]. The intervals show the minimum and maximum values obtained for each variable.

performs better in terms of tightness, however unlike our approach, its runtime distinctly increases with the number of iterations. Our implementation of LGG [38] using Convex Polyhedra with octahedral templates becomes slower than the abstractly accelerated version even for small time horizons (our implementation of LGG requires $\sim$4 ms for each iteration on a 6-dimensional problem with octahedral templates). Faster implementations of LGG will change the point at which the speed trade-off happens, but abstract acceleration will always be faster for long enough time horizons.

The evident advantage of abstract acceleration is its speed over finite horizons without much loss in precision, and of course the ability to prove properties for unbounded-time horizons.

9.3 Comparison with alternative abstract acceleration techniques

Table 6 shows a comparison between our approach and [49]. As can be seen, our approach is not only faster but much more precise. The reasons for this are many-fold. In terms of speed, the fact that [49] uses a dynamical model that is double the dimension of the one presented here and that the algorithmic complexity to manipulate it is $O(n^3)$ (n being the model dimension) will result in slower computations, even when using sparse matrices. In terms of precision, creating an over-approximation around the centre of the input set, as opposed to the origin, makes a considerable difference, which is increased by the fact that circular over-approximations are contained within the interval hulls used in [49], and are therefore up to $\frac{4}{3}^n$ times smaller in volume. This last consideration is only relevant for rotating dynamics, whereas positive real eigenvalues exhibit the same precision in both approaches. Finally, notice that in the ConvoyCar2 example, where all original eigenvalues are convergent, the interval hull approach [49] creates one divergent eigenvalue, which causes a critical change in the behaviour of the dynamics that leads to unbounded results.

9.4 Scalability

Finally, in terms of scalability, we have an expected $O(p^3)$ complexity w.r.t. the number of variables p, which derives from the simplex algorithm and the matrix multiplications in Equation (25). We have parametrised the number of cars in the convoyCar example [44] (also seen in Table 3), and experimented with up to 33 cars (each car after the first requires 3 state variables, so that for example we obtain $(33-1)\times 3 = 96$ variables for the 33-car

# of variables	3	6	12	24	48	96
runtime (sec)	0.004	0.031	0.062	0.477	5.4	56

Table 7 Scalability features of reachability computations via Abstract Acceleration.

case), using the same initial states for all cars. We report an average obtained from 10 runs for each configuration. These results demonstrate that our method can scale to industrial-size problems.

10 Related Work

There are several approaches that tackle the safety verification problem for dynamical models. The time-bounded analysis is in most cases unsound, since it cannot reason about unbounded-time cases (we note that a proof of the existence of a fix-point for the given horizon would restore such soundness by many tools do not attempt to find such proof which is left to the user). Unbounded-time solutions are therefore preferred when soundness is required, although they are often either less precise or slower than bounded counterparts.

10.1 Time-bounded reachability analysis

The first approach surrenders exhaustive analysis over the infinite time horizon, and restricts the exploration of model dynamics up to some given finite time bound. Decision procedures for bounded-time reachability problems have made much progress in the past decade, and computational algorithms with related software tools developed. Representatives are STRONG [24], HySon [12], CORA [2], HYLAA [5], Ariadne [7], and SpaceEx [30].

Set-based simulation methods generalise guaranteed integration of ODEs [11, 51], from enclosing intervals to relational domains. They use precise abstractions with low computational cost to over-approximate sets of reachable states up to a given time horizon. Early tools employed polyhedral sets (HyTech [41] and PHAVer [29]), polyhedral flow-pipes [18], ellipsoids [10], and zonotopes [34]. A breakthrough has been achieved by [35,38], with the representation of convex sets using template polyhedra and support functions. This method is implemented in the tool SpaceEx [30], which can handle dynamical systems with hundreds of variables. It performs computations using floating-point numbers, which is a deliberate choice to boost performance. On the other hand, although quite reasonable, its implementation is numerically unsound and therefore does not provide genuine formal guarantees, which is at the core of this work. More generally, most tools using eigendecomposition over a large number of variables (more than 10) happen to be numerically unsound due to the use of unchecked floating-point arithmetics. Some tools (e.g. C2E2 [25]) add a robustness check to the reachability computations in order to restore soundness by over-approximating simulation-based reachability analysis. Another breakthrough in performance has been achieved by HYLAA [5], which is the first tool to solve high-order problems, including several with hundreds of state variables. [7, 17] deal with non-linear dynamics (which are beyond the scope of this work), and the latter does so soundly. Other approaches focus on bounded model checking of hybrid automata, and use specialised constraint solvers (HySAT [28], iSAT [26]), or SMT encodings [19, 39].

10.2 Unbounded reachability analysis

The second approach, epitomised by static analysis methods [40], explores unbounded-time horizons. It employs conservative over-approximations to achieve relative completeness (this ensures that we always find a fixpoint) and decidability over infinite time horizons.

Unbounded techniques attempt to infer or compute a *loop invariant*, i.e., an inductive set that includes all reachable states. If the computed invariant is disjoint from the set of bad states, this proves that the latter are unreachable and hence that the loop is safe. However, analysers frequently struggle to obtain an invariant that is precise enough, with acceptable computational costs. The problem is evidently exacerbated by non-determinism in the loop, which corresponds to the case of open systems (i.e. with inputs). Prominent representatives of this analysis approach include Passel [45], Sting [21], and abstract interpreters such as Astrée [8] and InterProc [43]. Early work in this area has used implementations of abstract interpretation and widening [22], which represent still the underpinnings of many modern tools. The work in [40] uses abstract interpretation with convex polyhedra over piecewise differential inclusions. A more recent approach uses a CEGAR loop to improve the precision of polyhedral abstractions [9] following an inductive sequence of half-space interpolants. [23] employ optimisation-based (max-strategy iteration) with linear templates for hybrid systems with linear dynamics. Relational abstractions [54] use ad-hoc "loop summarisation" of flow relations, while abstract acceleration focuses on linear relations analysis [36, 37], which is common in program analysis.

10.3 Abstract acceleration

Abstract acceleration [36, 37, 44] captures the effect of an arbitrary number of loop iterations with a single, non-iterative transfer function that is applied to the entry state of the loop (i.e., to the set of initial conditions of the linear dynamics). Abstract acceleration has been extended from its original version to encompass inputs over reactive systems [56] but restricted to subclasses of linear loops, and later to general linear loops but without inputs [44].

The work presented in this article mitigates these limitations by presenting abstract acceleration for *general* linear loops *with* inputs [14, 15], developing numeric techniques for scalability and extending the domain to continuous time models.

The work in [50] puts forward improvements on [14], and its outcomes can be compared with the results in [15, 16]. Indeed, [50] proposes an alternative approach to Abstract Acceleration with inputs, which relies on the expansion of the dynamical equations to a model that is four times the original size (dimension). Although this model is mathematically more tractable, it is slower and more imprecise than the one presented in [15]. The reason for this is that while the latter uses a semi-circular over-approximation of the variable inputs, the former uses a rectangular over-approximation of the dynamics, which is in turn expanded by the acceleration. In particular, this rectangular over-approximation of the dynamics may cause a convergent model to become divergent, which leads to unbounded supports. There are cases in which this rectangular over-approximation can give a tighter result, especially if it is implemented using the symmetry properties described in [15], thus a combined approach may be more efficient. Despite the computational overhead, the handling of the guards presented in [50] has potential advantages over our approach: its description is formal, and indeed our procedure is in general not complete, resulting on occasion in an inability to find reasonably precise bounds. However, in a typical case our procedure works well and indeed in the experiments we have run, it has always been able to find the optimal solution.

11 Conclusions and Future Work

We have presented a thorough development of the Abstract Acceleration paradigm to guarded LTI models (namely, conditional linear loops) with inputs. We have extended existing work, which dealt only with autonomous models (i.e., models without inputs). We have decisively shown that the new approach over-competes state-of-the-art tools for unbounded-time reachability analysis in both precision and scalability. In particular, on the one hand we

claim full soundness (both algorithmic and numerical) of the proposed procedure and of its implementation. On the other, the new approach is capable of handling general unbounded-time safety analysis for large-scale open models.

Nested loops are out of the scope of this contribution, but represent relevant models that ought to be considered. Further work is also needed to extend the approach to non-linear dynamics, which we believe can be explored via hybridisation techniques [3]. We also plan to formalise the framework for general hybrid models, namely dynamical models with multiple discrete locations and location-dependent dynamics under multiple guards.

Acknowledgments

We would like to thank Colas Le Guernic for his constructive suggestions and comments on the paper. This work is supported by the Alan Turing Institute, London, UK, by EPSRC grant EP/J012564/1, ERC project 280053 (CPROVER), by the H2020 FET OPEN 712689 SC^2, and by Oxford Instruments PLC.

References

1. A. Adimoolam and T. Dang. Template complex zonotope based stability verification. In *Control Subject to Computational and Communication Constraints: Current Challenges*, pages 83–96. Springer, 2018.
2. M. Althoff. An introduction to cora 2015. In *ARCH@ CPSWeek*, pages 120–151, 2015.
3. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.
4. D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, 8(1):295–313, 1992.
5. S. Bak and P. S. Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, PA, USA, April 18-20, 2017*, pages 173–178, 2017.
6. S. Bennett. A brief history of automatic control. *IEEE Control Systems*, 16(3):17–25, 1996.
7. L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa. Assume-guarantee verification of nonlinear hybrid systems with Ariadne. *Int. J. Robust. Nonlinear Control*, 24(4):699–724, 2014.
8. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI*, pages 196–207. ACM, 2003.
9. S. Bogomolov, G. Frehse, M. Giacobbe, and T. A. Henzinger. Counterexample-guided refinement of template polyhedra. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 589–606. Springer, 2017.
10. O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *HSCC*, LNCS, pages 73–88. Springer, 2000.
11. O. Bouissou. *Analyse statique par interprétation abstraite de systèmes hybrides*. PhD thesis, École Polytechnique, 2008.
12. O. Bouissou, S. Mimram, and A. Chapoutot. Hyson: Set-based simulation of hybrid systems. In *Rapid System Prototyping (RSP), 2012 23rd IEEE International Symposium on*, pages 79–85. IEEE, 2012.
13. C. Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907.
14. D. Cattaruzza, A. Abate, P. Schrammel, and D. Kroening. Unbounded-time analysis of guarded LTI systems with inputs by abstract acceleration. In *SAS*, volume 9291 of *LNCS*, pages 312–331. Springer, 2015.
15. D. Cattaruzza, A. Abate, P. Schrammel, and D. Kroening. Unbounded-time analysis of guarded LTI systems with inputs by abstract acceleration (extended version). Technical report, University of Oxford, 2015. `http://arxiv.org/abs/1506.05607`.
16. D. Cattaruzza, A. Abate, P. Schrammel, and D. Kroening. Sound numerical computations in abstract acceleration. In *International Workshop on Numerical Software Verification*, pages 38–60. Springer, 2017.
17. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.
18. A. Chutinan and B. H. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *CDC*, pages 2089–2094. IEEE Computer Society, 1998.
19. A. Cimatti, S. Mover, and S. Tonetta. SMT-based verification of hybrid systems. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

20. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.
21. M. A. Colón, S. Sankaranarayanan, and H. B. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, pages 420–432. Springer, 2003.
22. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
23. T. Dang and T. M. Gawlitza. Template-based unbounded time verification of affine hybrid automata. In *APLAS*, LNCS, pages 34–49. Springer, 2011.
24. Y. Deng, A. Rajhans, and A. A. Julius. STRONG: A trajectory-based verification toolbox for hybrid systems. In *Quantitative Evaluation of Systems*, volume 8054 of *LNCS*, pages 165–168. Springer, 2013.
25. P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2e2: a verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 68–82. Springer, 2015.
26. A. Eggers, M. Fränzle, and C. Herde. SAT Modulo ODE: A direct SAT approach to hybrid systems. In *ATVA*, volume 5311 of *LNCS*, pages 171–185. Springer, 2008.
27. A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *HSCC*, pages 326–341. Springer, 2004.
28. M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.
29. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *HSCC*, volume 3414 of *LNCS*, pages 258–273. Springer, 2005.
30. G. Frehse, C. L. Guernic, A. Donzé, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *CAV*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011.
31. K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and computer science*, pages 91–111. Springer, 1996.
32. S. Gao, J. Avigad, and E. M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *Automated Reasoning*, pages 286–300. Springer, 2012.
33. P. K. Ghosh and K. V. Kumar. Support function representation of convex bodies, its application in geometric computing, and some related representations. *Computer Vision and Image Understanding*, 72:379–403, 1998.
34. A. Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.
35. A. Girard, C. L. Guernic, and O. Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *HSCC*, volume 3927 of *LNCS*, pages 257–271. Springer, 2006.
36. L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *SAS*, LNCS, pages 144–160. Springer, 2006.
37. L. Gonnord and P. Schrammel. Abstract acceleration in linear relation analysis. *Science of Computer Programming*, 93(Part B):125–153, 2014.
38. C. L. Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *CAV*, volume 5643 of *LNCS*, pages 540–554. Springer, 2009.
39. S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *CAV*, volume 5123 of *LNCS*, pages 190–203. Springer, 2008.
40. N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximations. In *SAS*, volume 864 of *LNCS*, pages 223–237. Springer, 1994.
41. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Journal on Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.
42. R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.
43. B. Jeannet. Interproc analyzer for recursive programs with numerical variables, 2010. `http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`.
44. B. Jeannet, P. Schrammel, and S. Sankaranarayanan. Abstract acceleration of general linear loops. In *POPL*, pages 529–540. ACM, 2014.
45. T. T. Johnson and S. Mitra. Passel: A verification tool for parameterized networks of hybrid automata, 2012. `https://publish.illinois.edu/passel-tool/`.
46. R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM (JACM)*, 33(4):808–821, 1986.
47. C. Knospe. Pid control. *IEEE Control Systems*, 26(1):30–31, 2006.
48. P. Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, 2nd edition, 1984.
49. C. Le Guernic. Toward a sound analysis of guarded LTI loops with inputs by abstract acceleration. In *International Static Analysis Symposium*, pages 192–211. Springer, 2017.
50. C. Le Guernic. Toward a Sound Analysis of Guarded LTI Loops with Inputs by Abstract Acceleration (extended version). working paper or preprint, June 2017.

51. R. Löhner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, Universität Karlsruhe, 1988.
52. J. Ouaknine and J. Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 366–379. Society for Industrial and Applied Mathematics, 2014.
53. R. Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51–3, pages 406–413. Cambridge University Press, 1955.
54. S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *CAV*, volume 6806 of *LNCS*, pages 686–702. Springer, 2011.
55. P. Schrammel. Unbounded-time reachability analysis of hybrid systems by abstract acceleration. In *Embedded Software*, pages 51–54. IEEE, 2015.
56. P. Schrammel and B. Jeannet. Applying abstract acceleration to (co-)Reachability analysis of reactive programs. *Journal of Symbolic Computation*, 47(12):1512–1532, 2012.
57. C. F. Van Loan. Matrix computations (johns hopkins studies in mathematical sciences), 1996.