Unbounded-Time Analysis of Guarded LTI Systems with Inputs by Abstract Acceleration*

Dario Cattaruzza, Alessandro Abate, Peter Schrammel, and Daniel Kroening

Department of Computer Science University of Oxford

Abstract. Linear Time Invariant (LTI) systems are ubiquitous in software systems and control applications. Unbounded-time reachability analysis that can cope with industrial-scale models with thousands of variables is needed. To tackle this general problem, we use *abstract acceleration*, a method for unbounded-time polyhedral reachability analysis for linear systems. Existing variants of the method are restricted to closed systems, i.e., dynamical models without inputs or non-determinism. In this paper, we present an extension of abstract acceleration to linear loops *with inputs*, which correspond to discrete-time LTI control systems, and further study the interaction with guard conditions. The new method relies on a relaxation of the solution of the linear dynamical equation that leads to a precise over-approximation of the set of reachable states, which are evaluated using support functions. In order to increase scalability, we use floating-point computations and ensure soundness by interval arithmetic. Our experiments show that performance increases by several orders of magnitude over alternative approaches in the literature. In turn, this tremendous gain allows us to improve on precision by computing more expensive abstractions. We outperform state-of-the-art tools for unbounded-time analysis of LTI system with inputs in speed as well as in precision.

1 Introduction

Linear loops are an ubiquitous programming template. Linear loops iterate over continuous variables, which are updated with a linear transformation. Linear loops may be guarded, i.e., terminate if a given linear condition holds. 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. 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: the problem is deciding reachability of certain "bad states", in which the system exhibits unsafe behaviour. Bad states may, in linear loops, be encompassed by guard assertions.

Reachability in linear programs, however, is a formidable challenge for automatic analysers: the problem is undecidable despite the restriction to linear transformations (i.e., linear dynamics) and linear guards. Broadly, there are two principal approaches

* This research was supported by Oxford Instruments PLC, by the ARTEMIS Joint Undertaking under grant agreement number 295311 (VeTeSS), by the ERC project 280053 (CPROVER), by the EC IAPP project 324432 (AMBI), and by the John Fell OUP Research Fund.

to safety problems. The first approach is to attempt to infer a *loop invariant*, i.e., an inductive set of states 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 cost. The problem is evidently exacerbated by the presence of non-determinism in the loop, which corresponds to the case of open systems. Prominent representatives of this analysis approach include Passel [30], Sting [7], and abstract interpreters such as ASTRÉE [2] and InterProc [28].

The second approach is to surrender exhaustive analysis over the infinite time horizon, and to restrict the exploration to system dynamics up to some given finite time bound. Bounded-time reachability is decidable, and decision procedures for the resulting satisfiability problem have made much progress in the past decade. The precision related to the bounded analysis is offset by the price of uncertainty: behaviours beyond the given time bound are not considered, and may thus violate a safety requirement. Representatives are STRONG [11] and SpaceEx [16].

The goal of this paper 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. We present a new approach for performing *abstract acceleration*. Abstract acceleration [21, 22, 29] 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). The key contribution of this paper is to lift the restriction of [29] to closed systems, and thus to allow for the presence of non-determinism.

We summarise next the contributions of this work:

1. We present a new technique to include inputs (non-determinism) in the abstract acceleration of general linear loops, thus overcoming its greatest limitation.
2. We introduce the use of support functions in complex spaces, in order to increase the precision of previous abstract acceleration methods.
3. By extending abstract acceleration and combining it with the use of support functions, we produce a time-unbounded reachability analysis that overcomes the main barrier of state-of-the-art techniques and tools for linear hybrid systems with inputs.
4. We employ floating point computations associated to bounded error quantification, to significantly increase the speed and scalability of previous abstract acceleration techniques, while retaining soundness.

Related work We review contributions within the two main perspectives in reachability analysis of hybrid systems, dealing respectively with bounded- and unbounded-time problems.

The first approach deals with bounded-time horizons: set-based simulation methods generalise guaranteed integration [32] 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 used polyhedral sets (HYTECH [26], PHAVER [15]), polyhedral flow-pipes [5], ellipsoids [3] and zonotopes [19]. A breakthrough has been achieved by [20, 23], with the representation of convex sets using template polyhedra and support functions. This method is implemented in the tool SPACEEX [16], which can handle dynamical systems with hundreds of variables.

It performs computations using floating-point numbers: this is a deliberate choice to boost performance, which, although quite reasonable, is implemented in a way that is unsound and that does not provide genuine formal guarantees. Other approaches use specialised constraint solvers (HySAT [14], iSAT [12]), or SMT encodings [6, 24] for bounded model checking of hybrid automata.

The second approach, epitomised in static analysis methods [25], explores unbounded-time horizons. It employs conservative over-approximations to achieve completeness and decidability over infinite time horizons. Early work in this area has used implementations of abstract interpretation and widening [8], which are still the foundations of most modern tools. The work in [25] uses abstract interpretation with convex polyhedra over piecewise-constant differential inclusions. [10] employs optimisation-based (max-strategy iteration) with linear templates for hybrid systems with linear dynamics. Relational abstractions [33] use ad-hoc "loop summarisation" of flow relations, whilst abstract acceleration focuses on linear relations analysis [21, 22], which is common in program analysis. Abstract acceleration has been extended from its original version to encompass inputs over reactive systems [35] but restricted to subclasses of linear loops, and later to general linear loops but without inputs [29]. This paper lifts these limitations by presenting abstract acceleration for *general* linear loops *with* inputs.

2 Preliminaries

Abstract acceleration [21, 22] is a key technique for the verification of programs with loops. The state of the art for this technique has reached the level where we can perform abstract acceleration of general linear loops without inputs [29], and of some subclasses of linear loops with inputs [34, 35], to an acceptable degree of precision. We develop an abstract acceleration technique for *general* linear loops *with bounded inputs*, whilst improving the precision and ease of computation, in order to overcome the negative effects caused on the over-approximation by the presence of bounded non-determinism.

2.1 Model Syntax

We are interested in loops expressed in the form:

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

where $\boldsymbol{x} \in \mathbb{R}^p$ are the state variables, $\psi := \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}$ 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 $\boldsymbol{A} \in R^{p \times p}$ and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$ are linear transformations characterising the dynamics of the system. In particular, the special instance where $\psi = \top$ (i.e., "while true") represents a time-unbounded loop with no guards, for which the discovery of a suitable invariant (when existing) is paramount. As evident at a semantical level (see next), 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 $G = \{\boldsymbol{x} \mid \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\}$.

2.2 Model Semantics

The traces of the model starting from an initial set $X_0 \subseteq \mathbb{R}^p$, with inputs restricted to $U \subseteq \mathbb{R}^q$, 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 $\forall n \geq 0, \boldsymbol{x}_{n+1} =$

$\tau(\boldsymbol{x}_n, \boldsymbol{u}_n)$, where

$$\tau(\boldsymbol{x}_n, \boldsymbol{u}_n) = (\boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{u}_n \mid \boldsymbol{G}\boldsymbol{x}_n \leq \boldsymbol{h} \wedge \boldsymbol{u}_n \in U) \,. \tag{1}$$

We extend the notation above to convex sets of initial conditions and inputs (X_0 and U), and write $\tau(X_0, U)$ to denote the set of states $\{\boldsymbol{x} \mid \boldsymbol{x}_0 \in X_0 \wedge \boldsymbol{u} \in U \wedge \boldsymbol{x} = \tau(\boldsymbol{x}_0, \boldsymbol{u})\}$ reached from X_0 by τ in one step. We furthermore write $\tau^n(X_0, U)$ to denote the set of states reached from X_0 via τ in n steps (*n-reach set*), i.e. for $n \geq 1$

$$\tau^n(X_0, U) = \{\boldsymbol{x}_n \mid \boldsymbol{x}_0 \in X_0 \wedge \forall k \in [0, n-1] : \boldsymbol{u}_k \in U \wedge \boldsymbol{x}_{k+1} = \tau(\boldsymbol{x}_k, \boldsymbol{u}_k)\} \,. \tag{2}$$

Since 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* $\hat{X}_n = \hat{\tau}^n(X_0, U) = \bigcup_{k\in[0,n]} \tau^k(X_0, U)$ as the union of the reachable sets over n iterations. Moreover, $\hat{X} = \bigcup_{n\geq 0} \tau^n(X_0, U)$ extends the previous notion over an unbounded time horizon.

2.3 Abstract Acceleration

Abstract Acceleration [22] is a method to over-approximate the *reach tube* of linear systems over any given time interval, including the infinite time horizon. [29] discusses this abstraction technique for systems without inputs, where an *abstract matrix* $\mathcal{A}^n$ is synthesised to encompass the combined dynamics generating all reach sets up to the n^{th} iteration. The abstract matrix $\mathcal{A}^n$ over-approximates the set of matrices $\bigcup_{k\in[0,n]} \boldsymbol{A}^k$. The reach tube $\hat{\tau}^n(X_0)$ (tailoring the notation above to a system *without* inputs) can then be over-approximated via the *abstract matrix multiplication* $\mathcal{A}^n X_0$ [29]. We will employ the notation $\mathcal{A}$ (rather than $\mathcal{A}^\infty$) to represent this notion over an infinite time horizon.

In this paper we extend this approach to systems with inputs, so that

$$\hat{\tau}^n(X_0, U) \subseteq \mathcal{A}^n X_0 \oplus \mathcal{B}^n U, \tag{3}$$

where $A \oplus B$ represents the Minkowski sum of two sets, namely $\{a + b \mid a \in A \wedge b \in B\}$, whereas the abstract matrix $\mathcal{B}^n$ over-approximates the set of matrices $\bigcup_{k\in[0,n]}(\boldsymbol{I}-\boldsymbol{A}^k)(\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}$, where $\boldsymbol{I}$ is a properly-sized identity matrix – this second approximation will be discussed in detail in Section 3.

2.4 Support Functions

There exist many abstract domains (namely, over-approximations) to encompass sets of states that are suitable for systems with linear dynamics, of which by far the most popular is that of *convex polyhedra* [9]. Rectangular abstractions are easy to process [36], but the over-approximations may be too conservative, which results in an even larger problem in the presence of non-deterministic inputs.

Abstract acceleration requires two abstract domains: the first to abstract the model dynamics – the original approach for abstract acceleration [29] uses *logahedra* [27] – and the second to represent spatial sets (convex polyhedra in [29]). In [29] the estimation of the number of loop iterations (time steps) leverages abstractions of initial sets as hypercubes, which is a source of imprecision that our method will not exhibit.

In this work, we use support functions [18, 31] for the abstract domains. Support functions have proven to be one of the most successful abstractions for the representation of reachability sets for dynamical and hybrid linear systems. A general assertion $\boldsymbol{Cx} \leq \boldsymbol{d}$ (of which the guard $\boldsymbol{Gx} \leq \boldsymbol{h}$ is just an example) entails a set of states that is a convex polyhedron, where each row in $\boldsymbol{C}$ is a direction orthogonal to a face in the polyhedron, and the corresponding value in $\boldsymbol{d}$ is the distance of that face to the origin.

Support functions represent a set by defining the distance of its convex hull with respect to a number of given directions. More specifically, the distance from the origin to the hyperplane that is orthogonal to the given direction and that touches its convex hull at its farthest. Finitely sampled support functions are template polyhedra in which the directions are not fixed, which helps avoiding wrapping effects [20]. The larger the number of directions provided, the more precisely represented the set will be. 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

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

where $< \boldsymbol{x}, \boldsymbol{v} >$ is the dot product of the two vectors.

Support functions do not exclusively apply to convex polyhedra, but in fact to any set $X \subseteq \mathbb{R}^p$ represented by a general assertion $\theta(X)$. We will restrict ourselves to the use of convex polyhedra, in which case the support function definition translates to solving the linear program

$$\rho_X(\boldsymbol{v}) = \max\{< \boldsymbol{x}, \boldsymbol{v} >| \boldsymbol{Cx} \leq \boldsymbol{d}\} . \tag{4}$$

Several properties of support functions allow us to reduce operational complexity. The most significant are [18]:

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

As can be seen by their structure, some of these properties reduce complexity to lower-order polynomial or even to constant time, by exchanging matrix-matrix multiplications ($O(p^3)$) into matrix-vector ($O(p^2)$), or into scalar multiplications.

3 Abstract Acceleration with Inputs

3.1 Overview of the Algorithm

Our algorithm takes as input the set of initial states X_0, the set of bounded inputs U, and the dynamics of a linear loop characterised by $\boldsymbol{G}, \boldsymbol{h}, \boldsymbol{A}$, and $\boldsymbol{B}$; and returns as output an over-approximation $\hat{X}^\sharp$ of the reach tube $\hat{X}$ (or corresponding quantities for the bounded-horizon case). We over- and under-approximate the number of loop iterations n that are required to first intersect and completely go beyond the guard set G, respectively, by means of the reach sets computed with the model dynamics: we denote these two quantities by $\underline{n}$ and $\overline{n}$. In the following we employ the notations $\sqcap$ for intersection of polyhedra, and $\sqcup$ for the convex hull $conv(X_1 \cup X_2)$.

If $\underline{n}$ or $\overline{n}$ are unbounded, we compute the abstract matrices $\mathcal{A}$ and $\mathcal{B}$ (as defined shortly), and return the quantity

$$\hat{X}^\sharp = X_0 \sqcup (\mathcal{A}(X_0 \sqcap G) \oplus \mathcal{B}U) \sqcap G \tag{5}$$

as the resulting reach tube, where again $G = \{\boldsymbol{x} \mid \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\}$. Otherwise, in the finite case, we compute the abstract matrices $\mathcal{A}^{\underline{n}}$ and $\mathcal{A}^{\overline{n}-\underline{n}}$ and set

$$\hat{X}_n^\sharp = X_0 \sqcup ((\mathcal{A}^{\underline{n}}(X_0 \sqcap G) \oplus \mathcal{B}^{\underline{n}}U) \sqcap G) \sqcup \left((\mathcal{A}^{\overline{n}-\underline{n}}(X_{\underline{n}} \sqcap G) \oplus (\mathcal{B}^{\overline{n}-\underline{n}}U)) \sqcap G\right) . \tag{6}$$

In this formula, the abstract matrices $\mathcal{A}^n$ and $\mathcal{B}^n$ are obtained as an over-approximation of sets of matrices, as described in Section 3.3.

3.2 Abstract Acceleration without Guards

With reference to (3), we now detail the abstract acceleration with inputs. Unfolding (2) we obtain

$$\boldsymbol{x}_n = \boldsymbol{A}^n\boldsymbol{x}_0 + \boldsymbol{A}^{n-1}\boldsymbol{B}\boldsymbol{u}_0 + \boldsymbol{A}^{n-2}\boldsymbol{B}\boldsymbol{u}_1 + ... + \boldsymbol{B}\boldsymbol{u}_{n-1} = \boldsymbol{A}^n\boldsymbol{x}_0 + \sum_{k\in[1,n]} \boldsymbol{A}^{n-k}\boldsymbol{B}\boldsymbol{u}_{k-1}.$$

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

$$\tau^\sharp(X_0, U) = \boldsymbol{A}(X_0 \cap G) \oplus \boldsymbol{B}U. \tag{7}$$

Then the reach set (as said, we ignore the presence of the guard set G for the time being) can be computed as

$$X_n = \boldsymbol{A}^n X_0 \oplus \boldsymbol{A}^{n-1}\boldsymbol{B}U + \boldsymbol{A}^{n-2}\boldsymbol{B}U \oplus ... \oplus \boldsymbol{B}U = \boldsymbol{A}^n X_0 \oplus \textstyle\sum_{k\in[0,n-1]} \boldsymbol{A}^k\boldsymbol{B}U.$$

What is left to do is to further simplify the sum $\sum_{k\in[0,n-1]} \boldsymbol{A}^k\boldsymbol{B}U$. We can exploit the following simple results from linear algebra.

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

It is evident that there are some restrictions on the nature of matrix $\boldsymbol{A}$: since we need to calculate the inverse $(\boldsymbol{I} - \boldsymbol{A})^{-1}$, $\boldsymbol{A}$ must not include the eigenvalue 1, i.e. $1 \notin \sigma(\boldsymbol{A})$, 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 eigen-decomposition of $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}$, and setting trivially $\boldsymbol{I} = \boldsymbol{S}\boldsymbol{I}\boldsymbol{S}^{-1}$, by the distributive and transitive property we obtain

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

While this does not directly eliminate the problem of the inverse for eigenvalues equal to 1, it allows us to set

$$\sum_{k=0}^{n-1} \lambda^k = \begin{cases} n & \lambda = 1 \\ \frac{1-\lambda^n}{1-\lambda} & \lambda \neq 1 \end{cases} \Rightarrow (\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1} = \boldsymbol{S}\ \mathit{diag}\begin{pmatrix} n & \lambda_i = 1 \\ \frac{1-\lambda_i^n}{1-\lambda_i} & \lambda_i \neq 1 \end{pmatrix} \boldsymbol{S}^{-1}. \tag{8}$$

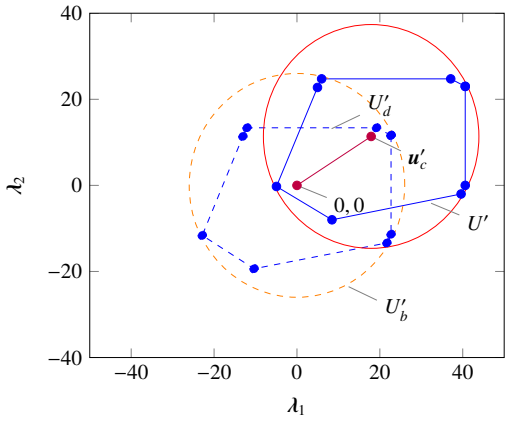

Fig. 1. Relaxation of an input set in a complex subspace making it invariant to matrix rotations. The dashed orange line is the red circle translated onto the origin.

In the case of Jordan blocks of size > 1, the entries in the k^{th} upper diagonal of the block are filled with the value: $\frac{-1^k}{k+1}\frac{1-\lambda^n}{(1-\lambda)^{k+1}} + \sum_{j=1}^{k}\frac{-1^{k-j}}{k-j}\binom{n}{j-1}\frac{\lambda^{n-j-1}}{(1-\lambda)^{k-j}}$.

This result can be only directly applied under restricted conditions, for instance whenever $\forall k > 0 : \boldsymbol{u}_k = \boldsymbol{u}_{k-1}$. In order to generalise it (in particular to non-constant inputs), we will over-approximate $\boldsymbol{B}U$ over the eigenspace by a spheral enclosure with centre $\boldsymbol{u}'_c$ and radius U'_b. To this end, we first rewrite

$$U'_J = \boldsymbol{S}^{-1}\boldsymbol{B}U = \{\boldsymbol{u}'_c\} \oplus U'_d \text{ , with } \boldsymbol{u}'_c[i] = \frac{1}{2}(\rho_{U'_J}(\boldsymbol{v_i}) + \rho_{U'_J}(-\boldsymbol{v_i})), \boldsymbol{v}_i[j] = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}$$

We then over-approximate U'_d via U'_b, by the maximum radius in the directions of the complex eigenvalues and non-singular Jordan blocks, as portrayed in Figure 1:

$$U'_b \supseteq U'_d \ : \ \forall i, j, \ \rho_{U'_b}(\boldsymbol{v}) = \begin{cases} max(\rho_{U'_d}(\boldsymbol{v'})) \text{ if } \lambda_i = \lambda_j^* \wedge |\boldsymbol{v'}| = |\boldsymbol{v}| \wedge (\boldsymbol{v'}[j] \neq 0 \vee \boldsymbol{v'}[i] \neq 0) \\ \rho_{U'_d}(\boldsymbol{v}) \qquad \text{otherwise} \end{cases}$$

Since the description of U'_b is no longer polyhedral, we will also create an image A_b of A that describes non-polyhedral faces in the directions of the complex eigenvectors ($\lambda_{bi} = |\lambda_i|$).

Returning to our original equation for the n-reach set, we obtain[1]

$$X_n \subseteq \boldsymbol{A}^n X_0 \oplus (\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}U_c \oplus (\boldsymbol{I} - \boldsymbol{A}_b^n)(\boldsymbol{I} - \boldsymbol{A}_b)^{-1}\boldsymbol{B}U_b, \text{ with } U_c = \{\boldsymbol{u}_c\} \quad (9)$$

Shifting the attention from reach sets to tubes, we can now over-approximate the *reach tube* by abstract acceleration of the three summands in (9), as follows.

Theorem 1. *The abstract acceleration* $\tau^{\sharp n}(X_0, U) =_{\text{def}} \mathcal{A}^n X_0 \oplus \mathcal{B}_c^n U_c \oplus \mathcal{B}_b^n U_b$ *is an over-approximation of the n-reach tube, namely* $\hat{X}_n \subseteq \tau^{\sharp n}(X_0, U)$.

We will discuss in the next section how to compute the abstract matrices $\mathcal{A}^n$, $\mathcal{B}_c^n$, and $\mathcal{B}_b^n$, with focus in particular on $\mathcal{A}^n$.

[1] Note that $\forall U'_b, U'_c, U'_d ; \exists U_b, U_c, U_d : U'_b = \boldsymbol{S}^{-1}BU_b$ so that $U'_c = \boldsymbol{S}^{-1}BU_c$ and $U'_d = \boldsymbol{S}^{-1}BU_d$. Hence, this inclusion is also valid in the original state space.

3.3 Computation of Abstract Matrices

We define the abstract matrix $\mathcal{A}^n$ as an over-approximation of the union of the powers of matrix $\boldsymbol{A}^k$: $\mathcal{A}^n \supseteq \bigcup_{k\in[0,n]} \boldsymbol{A}^k$. Next we explain how to compute such an abstract matrix. 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 3.5. Similar to [29], 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 eigenvectors. We can then easily compute $\boldsymbol{A}^n = \boldsymbol{S}\boldsymbol{J}^n\boldsymbol{S}^{-1}$, where

$$\boldsymbol{J}^n = \begin{bmatrix} \boldsymbol{J}_1^n & & \\ & \ddots & \\ & & \boldsymbol{J}_r^n \end{bmatrix}, \text{ with } \boldsymbol{J}_s^n = \begin{bmatrix} \lambda_s^n & \binom{n}{1}\lambda_s^{n-1} & \dots & \binom{n}{p_s-1}\lambda_s^{n-p_s+1} \\ & \lambda_s^n & \binom{n}{1}\lambda_s^{n-1} & \vdots \\ \vdots & & \ddots & \vdots \\ & & & \lambda_s^n \end{bmatrix} \text{ for } s \in [1, r]. \quad (10)$$

The abstract matrix $\mathcal{A}^n$ is computed as an abstraction over a vector $\boldsymbol{m}$ of non-constant entries of $\boldsymbol{J}^n$. The vector $\boldsymbol{m}$ is obtained by a transformation φ such that $\boldsymbol{J}^n = \varphi(\boldsymbol{m})$. If $\boldsymbol{J}^n$ is diagonal [29], then $\boldsymbol{m}$ equals the vector of powers of eigenvalues $(\lambda_0^n, \dots, \lambda_r^n)$. An interval abstraction can thus be simply obtained by computing the intervals $[\min\{\lambda_s^0, \lambda_s^n\}, \max\{\lambda_s^0, \lambda_s^n\}], s \in [1, r]$. We observe that the spectrum of the interval matrix $\sigma(\mathcal{A}^n)$ (defined as intuitively) is an over-approximation of $\bigcup_{k\in[0,n]} \sigma(\boldsymbol{A}^k)$.

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

Since the transformation φ transforms the vector $\boldsymbol{m}$ into the shape of (10) of $\boldsymbol{J}^n$, it is called a *matrix shape* [29]. We then define the abstract matrix as

$$\mathcal{A}^n = \{\boldsymbol{S}\ \varphi(\boldsymbol{m})\ \boldsymbol{S}^{-1} \mid \boldsymbol{\Phi}\boldsymbol{m} \leq \boldsymbol{f}\}\ , \quad (11)$$

where the constraint $\boldsymbol{\Phi}\boldsymbol{m} \leq \boldsymbol{f}$ 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 binomials) within $\boldsymbol{J}$, we find an over-approximation of the convex hull containing the points $\cup\{\boldsymbol{m}^k \mid 1\leq k\leq n\} \subseteq \{\boldsymbol{m} \mid \boldsymbol{\Phi}\boldsymbol{m}\leq\boldsymbol{f}\}$ with component-wise exponentiation $\boldsymbol{m}^k$.

As an improvement over [29], the rows in $\boldsymbol{\Phi}$ and $\boldsymbol{f}$ are synthesised by discovering support functions in these sets. The freedom of directions provided by these support functions results in an improvement over the logahedral abstractions used in previous papers (see Figure 2).

An additional drawback of [29] is that calculating the exact Jordan form of any matrix is computationally expensive and hard to achieve for large-dimensional matrices. We will instead use numerical algorithms in order to get an approximation of the Jordan normal form and account for numerical errors. In particular, if we examine the nature of (5)–(6), we find out that the numerical operations are not iterative, therefore the errors do not accumulate with time. We use properties of eigenvalues to relax $\boldsymbol{f}$ by finding the maximum error in the calculations that can be determined by computing the norm

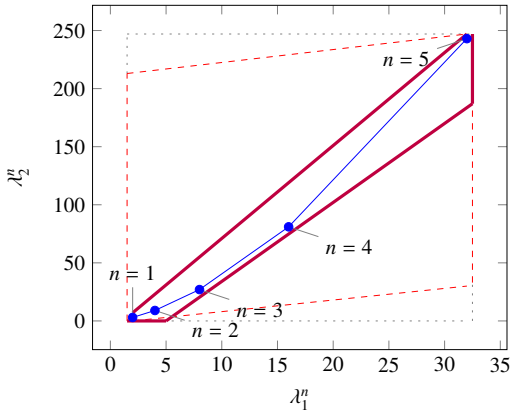

Fig. 2. Polyhedral faces from an $\mathbb{R}^2$ subspace, where $(\lambda_1^n, \lambda_2^n)$ so that λ_1=2, λ_2=3, $1 \leq n \leq 5$. Bold purple lines represent supports found by this paper. The dotted grey and dashed red polytopes show logahedral approximations (box and octagon) used in [29]. Note the scales (sloped dashed lines are parallel to the x=y line, and dashed red polytope hides two small sides yielding an octagon).

$\delta_{max} = |\boldsymbol{A} - \boldsymbol{S}\boldsymbol{J}_{est}\boldsymbol{S}^{-1}|$. The constraints $\boldsymbol{\Phi m} < \boldsymbol{f}$ are then computed by considering the ranges of eigenvalues $\lambda_s \pm \delta_{max}$ (represented in Fig. 2 as the diameter of the blue dots). The outward relaxation of the support functions ($\boldsymbol{f}$), which follows a principle similar to that introduced in [17], reduces the tightness of the over-approximation, but ensures the soundness of the abstract matrix $\mathcal{A}^n$ obtained. One can still use exact arithmetic with a noticeable improvement over previous work; however, for larger-scale systems the option of using floating point arithmetic, while taking into account errors and meticulously setting rounding modes, provides a 100-fold plus improvement that can make a difference towards rendering verification practically feasible.

The abstract matrices $\mathcal{B}_c^n$ and $\mathcal{B}_d^n$ (see Theorem 1), as well as $\mathcal{B}^n$, are defined similarly but using a similar assertion for the eigenvalues based on the transformations described in (8).

3.4 Abstract Acceleration with Guards: Estimation of the number of Iterations

The most important task remaining is how to calculate the number of iterations dealing with the presence of the guard set G.

Given a convex polyhedral guard expressed as the assertion $\{\boldsymbol{x} \mid \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\}$, 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 $\boldsymbol{g}_i = G_i^T$. The distance of the 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 of the hyperplane (for clarity, we assume the origin to be inside set X):

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

From the inequalities above we can determine up to which number of iterations 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:

$$\begin{aligned} &\rho_{X_0}(\boldsymbol{A}^{\underline{n_i}}\boldsymbol{g}_i) + \rho_{U'}((\boldsymbol{I} - \boldsymbol{A}^{\underline{n_i}})\boldsymbol{g}_i) \leq \gamma_i, \\ &\rho_{X_0}(-\boldsymbol{A}^{\overline{n_i}}\boldsymbol{g}_i) + \rho_{U'}((\boldsymbol{A}^{\overline{n_i}} - \boldsymbol{I})\boldsymbol{g}_i) \leq -\gamma_i. \end{aligned} \tag{12}$$

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 $\overline{n} = \min\{\ \overline{n_i}\ \}$.

Computing the maximum $\underline{n_i}$ such that (12) is satisfied is not easy, because the unknown $\underline{n_i}$ occurs in the exponent of the equation. However, if $\boldsymbol{g}_i$ was an eigenvector $\boldsymbol{v}_j$ of $\boldsymbol{A}$, we would have that $\boldsymbol{A}^{\underline{n_i}}\boldsymbol{v}_j = \lambda_j^{\underline{n_i}}\boldsymbol{v}_j$, which turns a p-dimensional problem into a 1-dimensional problem. However, since it is unlikely that the guards will be aligned to the eigenvectors, thus, we will use our support function properties to under- and over-approximate the number of iterations.

Let $\boldsymbol{g}_i = \sum_{j=1}^{p} a_{ij}\boldsymbol{v}_j + res(\boldsymbol{g}_i)$, where $\boldsymbol{v}_j$ are eigenvectors of $\boldsymbol{A}$, and $res(\boldsymbol{g}_i)$ is the component of $\boldsymbol{g}_i$ that lies outside the eigenspace of $\boldsymbol{A}$. Notice that this residual component will disappear during our calculations and is therefore not relevant. For simplicity we assume that all $a_{ij}\boldsymbol{v}_j$ are positive, extending the procedure for the general case through the development of the complex case in the Extended version. Then $\boldsymbol{A}^n\boldsymbol{g}_i = \sum_{j=1}^{p} \lambda_j^n a_{ij}\boldsymbol{v}_j + \boldsymbol{A}^n res(\boldsymbol{g}_i)$ where λ_j is the corresponding eigenvalue of $\boldsymbol{v}_j$. Since $res(\boldsymbol{g}_i)$ is orthogonal to the eigenspace of $\boldsymbol{A}$, then for $n > 0 : \boldsymbol{A}^n res(\boldsymbol{g}_i) = 0$, thus

$$\forall n > 0 : \boldsymbol{A}^n\boldsymbol{g}_i = \sum_{j=1}^{p} \lambda_j^n a_{ij}\boldsymbol{v}_j. \tag{13}$$

This way we can bound the first summand in (12) by $\rho_{X_0}(\boldsymbol{A}^n\boldsymbol{g}_i) \leq \sum_{j=1}^{p} \lambda_j^n a_{ij}\rho_{X_0}(\boldsymbol{v}_j)$. Using the support function properties detailed in Section 2.4, we obtain for (12):

$$\rho_{X_0}(A^n\boldsymbol{g}_i) + \rho_{U'}((I - A^n)\boldsymbol{g}_i) \leq \sum_{j=1}^{p} \lambda_j^n a_{ij}\rho_{X_0}(\boldsymbol{v}_j) + (\lambda_j^n a_{ij} - 1)\rho_{U'}(-\boldsymbol{v}_j) - \rho_{U'}(-res(\boldsymbol{g}_i)) \leq \gamma_i$$

In order to solve for n we transfer the constant terms to one side, taking into account that $\sum_{j=1}^{p} -\rho_{U'}(-\boldsymbol{v}_j) - \rho_{U'}(-res(\boldsymbol{g}_i)) = -\rho_{U'}(-\boldsymbol{g}_i)$, as

$$\sum_{j=1}^{p} \lambda_j^n a_{ij}(\rho_{X_0}(\boldsymbol{v}_j) + \rho_{U'}(-\boldsymbol{v}_j)) \leq \gamma_i + \rho_{U'}(-\boldsymbol{g}_i).$$

To separate the divergent element of the dynamics from the convergent one, let us define $b_{ij} = a_{ij}(\rho_{X_0}(\boldsymbol{v}_j) + \rho_{U'}(-\boldsymbol{v}_j))$ and $\lambda_m = max(\lambda_j)$ for all $j \in [1, p]$. Replacing, we obtain

$$\lambda_m^n \sum_{j=1}^{p} b_{ij}\left(\frac{\lambda_j}{\lambda_m}\right)^n \leq \gamma_i + \rho_{U'}(-\boldsymbol{g}_i)\ ,$$

which allows to finally formulate an iteration scheme for approximating n.

Proposition 1. *An iterative under-approximation of the number of iterations n can be computed by starting with* $\underline{n_i} = 0$ *and iterating over*

$$\underline{n_i} \geq \log_{\lambda_m}(\gamma_i + \rho_{U'}(-\boldsymbol{g}_i)) - \log_{\lambda_m}\left(\sum_{j=1}^{p} b_{ij}\left(\frac{\lambda_j}{\lambda_m}\right)^{\underline{n_i}}\right),$$

substituting the value of n_i *on the right-hand side and repeating a given number of times or up to convergence.*

In the case of $\overline{n_i}$ we must invert the eigenvectors and approximate from above, starting at a sufficiently large number (e.g. $\overline{n_i} = 10^{15}$), thus

$$\overline{n_i} \leq \log_{\lambda_m}(\gamma_i - \rho_{U'}(\boldsymbol{g}_i)) - \log_{\lambda_m}\left(\sum_{j=1}^{p} c_{ij}\left(\frac{\lambda_j}{\lambda_m}\right)^{\overline{n_i}}\right).$$

where $c_{ij} = a_{ij}(\rho_{X_0}(-\boldsymbol{v}_j) - \rho_{U'}(\boldsymbol{v}_j))$. If the initial $\overline{n_i}$ is not large enough, we simply double the exponent until the left hand side yields a smaller number than the one chosen originally.

3.5 Abstract Matrices for Complex Eigenvalues

To deal with complex numbers in eigenvalues and eigenvectors, [29] employs the real Jordan form for conjugate eigenvalues $\lambda = re^{i\theta}$ and $\lambda^* = re^{-i\theta}$ ($\theta \in [0, \pi]$), so that

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

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

In Section 3.3, in the case of real eigenvalues we have abstracted the entries in the power matrix $\boldsymbol{J}_s^n$ by ranges of eigenvalues $[\min\{\lambda_s^0, \lambda_s^n\}\ , \max\{\lambda_s^0, \lambda_s^n\}]$. In the complex case we can do something similar by rewriting eigenvalues into polar form $\lambda_s = r_s e^{i\theta_s}$ and abstracting by $[\min\{r_s^0, r_s^n\}\ ,\ \max\{r_s^0, r_s^n\}]e^{i[0\ ,\ \min(\theta_s, 2\pi)]}$.

What is left to do is to evaluate the effect of complex numbers on support functions: to the best of the authors' knowledge, there is no definition in the literature for support functions on complex numbers. We will therefore extend the manipulations for the real case directly to the complex one. For reasons of restricted space, please refer to extended version [4].

4 Case Study

We have selected a known benchmark to illustrate the discussed procedure: the room temperature control problem [13]. The temperature (variable `temp`) of a room is controlled to a user-defined set point (`set`), which can be changed at any time through a heating (`heat`) element, and is affected by ambient temperature (`amb`) that is out of the control of the system.

We formalise the description of such a system both via a linear loop and via hybrid dynamics. To begin with, observe that since such a system may be software controlled, we assume that part of the system is coded, and further assume that it is possible to discretise the physical environment for simulation. A pseudo-code fragment for the temperature control problem follows:

```
temp=5+read(35);
heat=read(1);
while(temp<400 && heat<300)
{
    amb=5+read(35);
    set=read(300);
    temp=.97 temp + .02 amb + .1 heat;
    heat=heat + .05(set-temp);
}
```

We use the `read` function to represent non-deterministic values between 0 and the maximum given as 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 $\delta_{max} < 10^{-17}$. Henceforth we focus on the latter description, as in the main text of this work. The eigen-decomposition of the dynamics is (the values are rounded to 3 decimal places):

$$A = SJS^{-1} \subseteq \begin{bmatrix} 0.798 & 0.173 \\ 0 & 0.577 \end{bmatrix} \begin{bmatrix} 0.985 \pm 10^{-16} & 0.069 \pm 10^{-17} \\ -0.069 \pm 10^{-17} & 0.985 \pm 10^{-16} \end{bmatrix} \begin{bmatrix} 1.253 & -0.376 \\ 0 & 1.732 \end{bmatrix}.$$

The discussed over-approximations of the reach-sets indicate that the temperature variable intersects the guard at iteration $\underline{n} = 32$. Considering the pseudo-eigenvalue matrix (described in the extended version for the case of complex eigenvalues) along these iterations, we use Equation (11) 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 (3), 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 system cooling (negative heating). The set is displayed in Figure 3, where for the sake of clarity we display only 8 directions of the 16 constraints. This results in a rather tight over-approximation that is not much looser than the convex hull of all reach sets obtained by [16] using the given directions. In Figure 3, we can see the initial set in black colour, the collection of reach sets in white, the convex hull of all reach sets in dark blue (as computed by [16]), and finally the abstractly accelerated set in light yellow (dashed lines). The outer lines represent the guards.

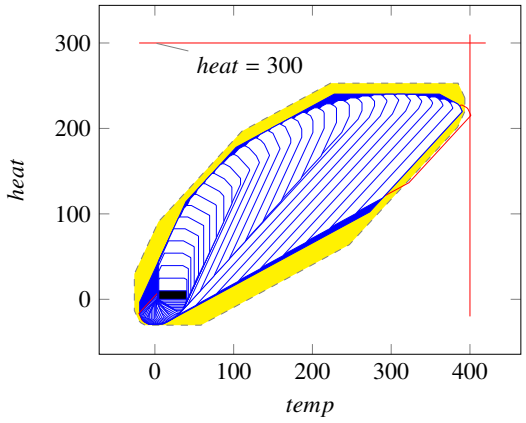

Fig. 3. The abstractly accelerated tube (yellow, dashed boundary), representing an over-approximation of the thermostat reach tube (dark blue). The set of initial conditions is shown in black, whereas successive reach sets are shown in white. The guards and the reach set that crosses them are close to the boundary in red.

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

type: s – stable loop, c – complex eigenvalues, g – loops with guard; **dim**: system dimension (variables); **bounds**: nb. of half-planes defining the polyhedral set;
IProc is [28]; **Sti** is [7]; **J+I** is this work;
improved: number of bounds newly detected by J+I over the existing tools (IProc, Sti)
Table 1. Experimental comparison of unbounded-time analysis tools with inputs

5 Implementation and Experimental Results

The algorithm has been implemented in C++ using the eigen-algebra package (v3.2), with double precision floating-point arithmetic, and has been tested on a 1.6 GHz core 2 duo computer.

Comparison with other unbounded-time approaches. In a first experiment we have benchmarked our implementation against the tools InterProc [28] and Sting [7]. We have tested these tools on different scenarios, including guarded/unguarded, stable/unstable and complex/real loops with inputs (details in Table 1).[2] It is important to note that in many instances, InterProc and Sting are unable to infer finite bounds at all.

[2] The benchmarks are available from `https://drive.google.com/file/d/0B22MA57MHHBKX2c3S05pT0d1OHc`.

	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 loop, c – complex eigenvalues, g – loops with guard; **dim**: system dimension (including fixed inputs); **bounds**: nb. of half-planes defining the polyhedral set; **improved**: number of bounds (and percentage) that were tighter (better) or looser (worse) than [29]; **J** is [29]; **mpfr+** is this paper using 1024bit mantissas ($e < 10^{-152}$); **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 Jordan form

Table 2. Experimental comparison with previous work

Table 2 shows the comparison of our implementation using different levels of precision (long double, 256 bit, and 1024 bit floating point precision) with the original abstract acceleration for linear loops without inputs (J) [29] (where inputs are fixed to constants). This shows that our implementation gives tighter over-approximations on most benchmarks (column 'improved'). Whilst on a limited number of instances the current implementation is less precise (Fig. 2 gives a hint why this is happening), the overall increased precision is owed to lifting the limitation on directions caused by the use of logahedral abstractions.

At the same time, our implementation is faster – even when used with 1024 bit floating point precision – than the original abstract acceleration (using rationals). 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 [29] (using the ATLAS package), whereas our implementation requires at most one second. For the last two benchmarks, the polyhedral computations blow up in [29], whereas our support function approach shows only moderately increasing runtimes. The increase of speed is owed to multiple factors, as detailed in Table 3. The difference of using long double precision floating points vs arbitrary precision arithmetic is negligible as all results in the given examples match exactly to 9 decimal places. Note that, as explained above, soundness can be ensured by correct rounding in the floating point computations.

Comparison with bounded-time approaches. In a third experiment, we compare our method with the LGG algorithm [23] used by SpaceEx [16]. In order to set up a fair comparison we have provided the implementation of the native algorithm in [23]. We have run both methods on the convoyCar example [29] with inputs, which presents an

unguarded, scalable, stable loop with complex dynamics, and focused on octahedral abstractions. For convex reach sets, the approximations computed by abstract acceleration are quite tight in comparison to those computed by the LGG algorithm. However, storing finite disjunctions of convex polyhedra, the LGG algorithm is able to generate non-convex reach tubes, which are arguably more proper in case of oscillating or spiralling dynamics. Still, in many applications abstract acceleration can provide a tight over-approximation of the convex hull of those non-convex reach sets.

Table 4 shows the results of this comparison. For simplicity, we present only the projection of the bounds along the variables of interest. As expected, the LGG algorithm performs better in terms of tightness, but its runtime increases with the number of iterations. Our implementation of LGG using Convex Polyhedra with octagonal templates is 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 octagonal abstraction). This can be improved by the use of zonotopes, or by careful selection of the directions along the eigenvectors, but this comes at a cost on precision. Even when finding combinations that outperform our approach, this will only allow the time horizon of the LGG approach to be slightly extended before matching the analysis time from abstract acceleration, and the reachable states will still remain unknown beyond the extended time horizon.

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

Scalability. Finally, in terms of scalability, we have an expected $O(n^3)$ complexity worst-case bound (from the matrix multiplications in equation 3). We have parameterised the number of cars in the convoyCar example [29] (also seen in Table 2), and experimented with up to 33 cars (each car after the first requires 3 variables, so that for example $(33-1)\times 3 = 96$ variables), and have adjusted the initial states/inputs sets. We report an average of 10 runs for each configuration. These results demonstrate that our method scales to industrial-size problems.

# of variables	3	6	12	24	48	96
runtime	4 ms	31 ms	62 ms	477 ms	5.4 s	56 s

6 Conclusions and Future Work

We have presented an extension of the Abstract Acceleration paradigm to guarded LTI systems (linear loops) with inputs, overcoming the limitations of existing work dealing with closed systems. We have decisively shown the new approach to over-compete

Optimization	Speed-up
Eigen vs. ATLAS (http://eigen.tuxfamily.org/index.php?title=Benchmark)	2–10
Support functions vs. generators for abstract matrix synthesis	2–40
long double vs. multiple precision arithmetic	5–200
Total	20–80000

Table 3. Performance improvements by feature

	this paper		LGG		
name	100 iterations	unbounded	100 iterations	200 iterations	300 iterations
run time	5 ms	5 ms	50 ms	140 ms	195 ms
car acceleration	[-0.895 1.34]	[-1.038 1.34]	[-0.802 1.31]	[-0.968 1.31]	[-0.968 1.31]
car speed	[-1.342 5.27]	[-4.059 5.27]	[-1.331 4.98]	[-3.651 4.98]	[-3.677 4.98]
car position	[42.66 83.8]	[42.66 90.3]	[43.32 95.5]	[43.32 95.5]	[43.32 95.6]

Table 4. Comparison on convoyCar2 benchmark, between this work and the LGG algorithm [23]

state-of-the-art tools for unbounded-time reachability analysis in both precision and scalability. The new approach is capable of handling general unbounded-time safety analysis for large scale open systems with reasonable precision and fast computation times. Conditionals inside loops and nested loops are out of the scope of this paper.

Work to be done is extending the approach to non-linear dynamics, which we believe can be explored via hybridisation techniques [1], and to formalise the framework for general hybrid models with multiple guards and location-dependent dynamics, with the aim to accelerate transitions across guards rather than integrate individual accelerations on either side of the guards.

References

1. Asarin, E., Dang, T., Girard, A.: Hybridization methods for the analysis of nonlinear systems. Acta Informatica 43(7), 451–476 (2007)
2. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: PLDI. pp. 196–207. ACM (2003)
3. Botchkarev, O., Tripakis, S.: Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In: HSCC. pp. 73–88. LNCS, Springer (2000)
4. Cattaruzza, D., Abate, A., Schrammel, P., Kroening, D.: Unbounded-time analysis of guarded lti systems with inputs by abstract acceleration (extended version). Tech. rep., University of Oxford (2015), `http://arxiv.org/abs/1506.05607`
5. Chutinan, A., Krogh, B.H.: Computing polyhedral approximations to flow pipes for dynamic systems. In: CDC. pp. 2089–2094. IEEE Computer Society (1998)
6. Cimatti, A., Mover, S., Tonetta, S.: SMT-based verification of hybrid systems. In: AAAI Conference on Artificial Intelligence. AAAI Press (2012)
7. Colón, M.A., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: CAV. pp. 420–432. Springer (2003)
8. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL. pp. 238–252 (1977)
9. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL. pp. 84–97. ACM (1978)
10. Dang, T., Gawlitza, T.M.: Template-based unbounded time verification of affine hybrid automata. In: APLAS. pp. 34–49. LNCS, Springer (2011)
11. Deng, Y., Rajhans, A., Julius, A.A.: STRONG: A trajectory-based verification toolbox for hybrid systems. In: Quantitative Evaluation of Systems. LNCS, vol. 8054, pp. 165–168. Springer (2013)
12. Eggers, A., Fränzle, M., Herde, C.: SAT Modulo ODE: A direct SAT approach to hybrid systems. In: ATVA. LNCS, vol. 5311, pp. 171–185. Springer (2008)
13. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: HSCC. pp. 326–341. Springer (2004)

14. Fränzle, M., Herde, C.: HySAT: An efficient proof engine for bounded model checking of hybrid systems. Formal Methods in System Design 30(3), 179–198 (2007)
15. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: HSCC. LNCS, vol. 3414, pp. 258–273. Springer (2005)
16. Frehse, G., Guernic, C.L., Donzé, A., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: CAV. LNCS, vol. 6806, pp. 379–395. Springer (2011)
17. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: Automated Reasoning, pp. 286–300. Springer (2012)
18. Ghosh, P.K., Kumar, K.V.: Support function representation of convex bodies, its application in geometric computing, and some related representations. Computer Vision and Image Understanding 72, 379–403 (1998)
19. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: HSCC. LNCS, vol. 3414, pp. 291–305. Springer (2005)
20. Girard, A., Guernic, C.L., Maler, O.: Efficient computation of reachable sets of linear time-invariant systems with inputs. In: HSCC. LNCS, vol. 3927, pp. 257–271. Springer (2006)
21. Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: SAS. pp. 144–160. LNCS, Springer (2006)
22. Gonnord, L., Schrammel, P.: Abstract acceleration in linear relation analysis. Science of Computer Programming 93(Part B), 125–153 (2014)
23. Guernic, C.L., Girard, A.: Reachability analysis of hybrid systems using support functions. In: CAV. LNCS, vol. 5643, pp. 540–554. Springer (2009)
24. Gulwani, S., Tiwari, A.: Constraint-based approach for analysis of hybrid systems. In: CAV. LNCS, vol. 5123, pp. 190–203. Springer (2008)
25. Halbwachs, N., Raymond, P., Proy, Y.E.: Verification of linear hybrid systems by means of convex approximations. In: SAS. LNCS, vol. 864, pp. 223–237. Springer (1994)
26. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. Journal on Software Tools for Technology Transfer 1(1-2), 110–122 (1997)
27. Howe, J.M., King, A.: Logahedra: A new weakly relational domain. In: ATVA, pp. 306–320. Springer (2009)
28. Jeannet, B.: Interproc analyzer for recursive programs with numerical variables (2010), `http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`
29. Jeannet, B., Schrammel, P., Sankaranarayanan, S.: Abstract acceleration of general linear loops. In: POPL. pp. 529–540. ACM (2014)
30. Johnson, T.T., Mitra, S.: Passel: A verification tool for parameterized networks of hybrid automata (2012), `https://publish.illinois.edu/passel-tool/`
31. Le Guernic, C.: Reachability analysis of hybrid systems with linear continuous dynamics. Univerit Joseph Fourier (2009)
32. Löhner, R.: Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen. Ph.D. thesis, Universität Karlsruhe (1988)
33. Sankaranarayanan, S., Tiwari, A.: Relational abstractions for continuous and hybrid systems. In: CAV. LNCS, vol. 6806, pp. 686–702. Springer (2011)
34. Schrammel, P., Jeannet, B.: Extending abstract acceleration to data-flow programs with numerical inputs. In: Numerical and Symbolic Abstract Domains. ENTCS, vol. 267, pp. 101–114. Elsevier (2010)
35. Schrammel, P., Jeannet, B.: Applying abstract acceleration to (co-)reachability analysis of reactive programs. Journal of Symbolic Computation 47(12), 1512–1532 (2012)
36. Stursberg, O., Krogh, B.H.: Efficient representation and computation of reachable sets for hybrid systems. In: HSCC. LNCS, vol. 2623, pp. 482–497. Springer (2003)