Verification and Synthesis of Linear Systems by Abstract Acceleration

Dario Cattaruzza

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2018

Abstract

Embedded systems are constantly growing in number and complexity. A large number of these relate to physical elements that have behaviour that is either linear or can be described using linear differential equations (e.g., vehicle speed/position, temperature control, oscillators, etc). In this work we explore the application of formal methods for safety checking and controller synthesis in the particular case of Linear Time Invariant (LTI) models where the dynamics may apply to both continuous and discrete variables in both continuous and discrete time. Our work applies to each of these cases independently. To this end, we use abstract acceleration, a method that combines abstract interpretation with acceleration in order to compute precise fix-points for the reach space of the model. Existing techniques have proven to be useful in the verification of discrete time systems, and we extend the method to a wider set of models and improve its performance. Furthermore, by applying control theory and SAT solving techniques, we explore the synthesis of correct by construction digital controllers using abstract acceleration as a model template. Our results show that the technique scales to models with several dozen variables for which sound results can be found in a matter of minutes.

Contents

Chapter 1

Introduction

1.1 Motivation

Linear loops are an ubiquitous programming pattern [115]. They iterate over continuous variables (in the case of physical systems) or discrete variables (in the case of program loops), which are updated with a linear transformation. Linear loops may be guarded, i.e., terminate if a given linear condition holds (at which point a system may either describe a new behaviour or simply halt). 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 [24, 115]. The usage of such models in safety-critical embedded systems makes linear loops a fundamental target for formal methods. In particular, cyber-physical control systems have a long history in design and implementation [24, 115] where well known failures may be avoided through the use of formal verification (e.g. overflow errors, overheating errors).

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. Modern implementations of embedded control systems have proliferated with the availability of low-cost devices

that can perform highly non-trivial control tasks, with significant impact in numerous application areas such as environmental control and robotics [19, 80]. Correct and sound control is non-trivial, however. Thus, the programming is a key barrier to broad adoption of digital control, and requires considerable expertise.

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 [112], and for low order models [136]). The problem is decidable when reduced to finite state spaces, but even then, algorithms are costly (in particular for the infinite time case). Existing work in this area provides tools aiding engineers to correctly design formally safe systems, but these tools often fall short in an industrial environment due to either their narrow scope or their lack of scalability.

In this work we seek to provide a tool that can help control engineers design formally verified, correct-by-construction, scalable linear embedded systems. We leverage existing techniques in the fields of formal verification, control theory and hybrid systems in order to achieve fast, sound algorithms for the verification and synthesis of embedded control systems.

1.2 Thesis

We claim that Abstract Acceleration (AA) is a scalable (for dozens of variables) and sound technique for 1) formal verification of safety properties of practical linear continuous- and discrete-time invariant (LTI) systems with inputs, and for 2) synthesis of correct-by-construction linear stabilizing controllers for these systems that meet given safety properties.

1.3 Dissertation structure and methodology

Our methodology starts by studying related theories in this domain and selecting a subset of these to apply to abstract acceleration in order to meet our targets. The relevant theories that we have selected are detailed in Chapter 2. Other methods that are used to achieve similar results are described in Chapter 3, where we also indicate the advantages (and disadvantages) and novelty of our technique w.r.t. the existing literature. In particular we look at the case of existing AA methods which are most relevant to our work. We dedicate Chapter 4 to explain all the additions we have made to the AA framework. We start by extending the current state of the art for abstract acceleration to continuous LTI models with variable inputs (before this work the technique only applied to discrete time systems with parametric inputs). This is previously unexplored theory, which lifts several limitations of abstract acceleration. Next, we address the matter of precision. Although existing AA techniques compare well with static analysis tools in this respect, their precision is not as good as that observed in the bounded model verification tools we have examined. We improve precision first by using a richer domain in the description of abstract matrices, and secondly by introducing abstraction refinement techniques. This is also discussed in Chapter 4.

The next necessary development relates to scalability. Prior to this work, abstract acceleration was only capable of dealing with systems up to 8-10 variables due to the high processing cost of symbolic analysis. In order to increase scalability (to dozens of variables in a few seconds with average polynomial performance), we develop numerical techniques and the necessary theory, while retaining soundness. This allows us to achieve results that will perform in similar times on systems ten times larger. The numeric techniques developed to achieve scalability while preserving soundness are discussed in Chapter 6.

In order to validate our thesis we use both theoretical and experimental results. The former shown in this dissertation in the form of theorems and proofs that demonstrate the improvements made on AA and how they impact our results, whereas the latter consists of an experimental evaluation of an implementation of our theoretical work in a software tool called Axelerator[1], which compares its performance against existing techniques.

[1] `www.cprover.org/LTI`

The second part of our work is dedicated to controller synthesis. In this case, we change our objective from validating an existing model to that of creating a model that meets a set of specifications. We focus on parametric models, where the structure of the model is given and we seek to synthesise values (also known as parameters) that make such a model meet the requirements. Synthesis is a more complex task than verification since it automates a larger portion of the design chain.

In Chapter 5 we use counterexample guided synthesis (CEGIS) to synthesise controllers for LTI plants. The objective of this section is to show the applicability of abstract acceleration to numerous objectives that involve more complex properties and which solve real world problems. Our CEGIS approach uses a new paradigm based on optimisation refinement. Contributions to the CEGIS infrastructure are detailed in this chapter. Once again, we implement our theoretical development into a tool called DSynth, which uses Axelerator as one of its back ends in order to synthesise correct-by-construction controllers. Results have been compared against a 'naive approach' to CEGIS-based synthesis in order to show the advantages of our technique.

The conclusions show the overall result of this comprehensive technique, the tool we developed to implement it and future directions for enhancing them.

1.4 Contributions

We next present a list of contributions made in this work which are new to the field. Some of these will be discussed with respect to existing work in Chapter 3.

On abstract acceleration

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 the use 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 employ floating-point computations associated to bounded error quantification, to significantly increase the speed and scalability of previous abstract acceleration techniques, while retaining soundness.

On controller synthesis

1. We automatically generate *correct-by-construction* digital state-feedback controllers that guarantee a given safety property alongside stability and response time specifications. We extend this to observer design. Observers are much more complex and difficult to synthesise than standard full state feedback controllers.

2. Our synthesis addresses challenges created by digitalisation errors, such as quantisation, and time discretisation, as well as delays in the sampling and processing times, all of which introduce control errors in real implementations.

3. Using abstract acceleration in combination with an error model for quantization, we present a method for soundly evaluating reachability in a combined continuous-discrete time/space.

4. We develop a model for inductive synthesis that combines CEGIS with abstraction refinement and optimisation in order to handle complex systems.

On decision procedures

1. We develop a numerical Simplex with error bounds using interval representations that ensures results are sound over-approximations of the original problem, i.e., the true results are always contained in the intervals. We note that, unlike previous work on sound linear solvers, our algorithm can reason about problems with pre-existing bounded errors in the problem statement.

2. We develop a Vertex Enumerator using intervals that soundly enumerates all possible vertices of a polyhedron (see Section 2.4.3) which has been described using imprecise arithmetics.

3. We develop a fast algorithm for calculating error bounds on numerically calculated eigenspaces in order to restore soundness. While this can largely be achieved using existing tools, the need to integrate multiple precision numbers with interval arithmetics and to deal with non-diagonalizable matrices while maintaining a reasonable speed has motivated us to apply a number of over-approximations that deliver fatser results without a great loss in precision.

Chapter 2

Preliminaries

2.1 Notation

Throughout this work the following notations will be used

- $\boldsymbol{x}$ — a bold lower case letter represents a vector. The dimensions of the vector will be given when introduced.
- $\boldsymbol{A}$ — a bold capital letter represents a matrix. The dimensions of the matrix will be given when introduced.
- $\mathbb{x}$ — a blackboard bold letter represents an interval vector. These will be described in the interval arithmetics section.
- $\mathbb{A}$ — a blackboard bold capital letter represents an interval matrix. These will be described in the interval arithmetics section.
- x — a plain letter represents an element without a specific type. This means it can be a vector, a set, or any other type of element.
- X — a plain capital letter represents a set. This is a set of elements which may be of any type, including matrices, vectors, etc.

- $\mathcal{A}$ — a calligraphic capital letter represents a set in an abstract domain (discussed in Section 2.4).
- $[1, \ldots, n]$ — a comma and ellipsis separated interval represents a set of integers (typically used for index ranges).
- $[0, T]$ — a comma separated interval represents an interval of reals. These will be described in Section 2.3.2.
- $\underline{x}$ and $\overline{x}$ represent a lower and an upper bound respectively (e.g. $[\underline{x}, \overline{x}]$).
- x^* corresponds to the complex conjugate, i.e. $(a + bi)^* = a - bi$. This applies element-wise to matrices and vectors.
- The notation $x \models \phi$ indicates that x satisfies property ϕ.
- The notation $\{x \mid x \models \phi\}$ indicates set comprehension, meaning the set of all x that satisfy ϕ.
- The notation $\boldsymbol{M}_{i,*}$ represents the i^{th} row of a matrix and $\boldsymbol{M}_{*,j}$ the j^{th} column.
- The notation $\boldsymbol{M}_i$ represents a matrix in a collection of matrices. In the case of vectors this may either represent a vector in a collection or the i^{th} element of the vector, which should be evident by the context of the formula. In some cases nested sub-indices may also be used, for example, $\boldsymbol{x}_{ij}$ represents the j^{th} element of the i^{th} vector.
- $\|\boldsymbol{A}\|_n = \left(\sum_i \sum_j \boldsymbol{A}_{ij}^n \right)^{\frac{1}{n}}$ represents the $n-$norm of a matrix (similarly for vectors).
- $|x|$ is the absolute value of x.
- $\dot{x} = \frac{dx}{dt}$ is the first derivative of x. i.e. the change of x w.r.t. to time.
- we use $[\ a_1 \quad a_2 \quad \cdots \quad a_p\ ]$ to represent a row vector and $[\cdots]^\intercal$ to represent its transpose.

2.2 Modelling systems

We are interested in system behaviour over time. In the case of safety, this corresponds to evaluating the state of the system at different times and verifying that they meet our properties. Since it is often unfeasible to perform such verification on a real system by directly measuring its behaviour, we use mathematical models that represent the behaviour of the system. These mathematical models are not always exact (see section on Abstraction), but are meant to represent the actual system with a certain degree of accuracy and can be made to preserve some properties. Since we are evaluating systems on both the physical and the digital world, we need different models that address the conditions of each domain. We will later show that it is possible for some of these models to represent others, thus allowing us to obtain a common framework for the combined behaviour of such systems. We note however, that our approach reasons over these models and it can only be as accurate as the model permits. Imprecision or unsoundness in the model will transfer to the results. Since this does not form part of the scope of this work, we will assume that the model is an accurate representation of the true system. In the case of a digital control system, we will refer to the physical system exhibiting continuous dynamics as the plant, and the digital system that controls it, which exhibits discrete dynamics, as the controller.

A linear time invariant (LTI) model is defined as a model whose state progression in time is expressed by linear dynamics that do not vary in time. This typically means that it can be expressed as a set of matrix multiplications applied directly over the variables. The traces of an LTI model do not always display linear behaviour, as the linear dynamics can be applied to derivatives of the variables, and become non-linear after integration. We will describe next our semantics for such models first for discrete time models and later for continuous ones, finally integrating both into a single framework.

2.2.1 Model properties

The properties we are interested in are characteristics of the model whose valuation can be verified against a specification. Typically, they do not represent direct measurements

of the state of the system, but traits that can be derived from such observations. In this work, we mainly focus on the following properties:

- Safety

  Safety is the ability of the model to avoid a set of bad states. The definition of bad states varies depending on the chosen models. In our case they are represented by exclusion as the set of states not contained in a given safe region. The way in which we describe these sets will be indicated later in this chapter.

- Stability

  The ability of a model to remain within a bounded set of states is called stability. There are different types of stability, such as Bounded Input/Bounded Output (BIBO) in which the requirement is that, given a bounded input, the output remains within certain limits, or asymptotic stability in which the output is expected to converge to a certain value. In our model, stability is directly related to convergence which will be discussed in Section 2.2.4.

- Settling time

  Settling time is a successor of stability. It indicates how fast the model becomes stable typically by establishing a margin around the target state within which the model is considered to have reached stability.

2.2.2 Linear loops with inputs

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

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

where $\boldsymbol{x} \in \mathbb{R}^p$ 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 $G = \{\boldsymbol{x} \mid \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\}$. 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.

2.2.3 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

$$\tau(\boldsymbol{x}_k, \boldsymbol{u}_k) = (\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \mid \boldsymbol{G}\boldsymbol{x}_k \leq \boldsymbol{h})\,. \tag{2.1}$$

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

$$\tau(X_k, U) = \{\tau(\boldsymbol{x}_k, \boldsymbol{u}_k) \mid \boldsymbol{x}_k \in X_k, \boldsymbol{u}_k \in U\}\,. \tag{2.2}$$

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

$$\tau^0(X_0, U) = X_0, \quad \tau^n(X_0, U) = \tau(\tau^{n-1}(X_0, U) \cap G, U). \tag{2.3}$$

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*

$$\hat{X}_n = \hat{\tau}^n(X_0, U) = \bigcup_{k\in[0,\dots,n]} \tau^k(X_0, U) \tag{2.4}$$

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).

2.2.4 Spectral eigendecomposition

Eigendecomposition [162] is a factorisation of a matrix into a canonical form that is characterised by having non-zero elements (the eigenvalues) only on the main diagonal (note that not all matrices can be factorised this way). 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, 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

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_1 & & \\ & \ddots & \\ & & \boldsymbol{J}_r \end{bmatrix} \text{ and } \boldsymbol{J}_{s\in[1,\dots,r]} = \begin{bmatrix} \lambda_s & 1 & \dots & 0 \\ 0 & \lambda_s & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & \lambda_s \end{bmatrix}. \tag{2.5}$$

In the case of the Jordan Form, the repeated eigenvectors are replaced by generalised eigenvectors, which have the property that $(\boldsymbol{A} - \lambda_s \boldsymbol{I})^j \boldsymbol{v}_j = 0$, where $\boldsymbol{v}_j$ is the j^{th} generalised eigenvector related to λ_s.

Finally, the eigenvalues of a real matrix may be complex numbers, which is inconvenient in the ensuing analysis. Rather than using complex arithmetic on these numbers, we choose a different representation denominated the pseudo-eigenspace. Pseudo-eigendecomposition relies on the observation that complex eigenvalues of a real matrix

always come in conjugate pairs. By relaxing the restriction of non-zero values from the main diagonal to include the immediate upper and lower diagonals next to it, we can perform the following equivalence:

$$\begin{aligned}\begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_{i+1} \end{bmatrix}\begin{bmatrix} \lambda_i & 0 \\ 0 & \lambda_{i+1} \end{bmatrix}\begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_{i+1} \end{bmatrix}^{-1} &= \begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_i^* \end{bmatrix}\begin{bmatrix} re^{\theta i} & 0 \\ 0 & re^{-\theta i} \end{bmatrix}\begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{v}_i^* \end{bmatrix}^{-1} \\ &= \begin{bmatrix} re(\boldsymbol{v}_i) & im(\boldsymbol{v}_i) \end{bmatrix}\begin{bmatrix} r\cos(\theta) & r\sin(\theta) \\ -r\sin(\theta) & r\cos(\theta) \end{bmatrix}\begin{bmatrix} re(\boldsymbol{v}_i) & im(\boldsymbol{v}_i) \end{bmatrix}^{-1}, \quad (2.6)\end{aligned}$$

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 this conversion is then performed. This representation is also called the Real Jordan Form.

2.2.5 Linear time invariant dynamical systems

The semantics described so far relate to discrete time models, where the passage of time is quantised. However, we are also interested in models that describe continuous time behaviour, hence we need to discuss the semantics of these and their relation to discrete time models. A dynamical system is a system in which a function describes the progression of a state over time. In a continuous domain with linear dynamics, an LTI system is often described by a first order Ordinary Differential Equation (ODE).

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t). \quad (2.7)$$

where $t \in \mathbb{R}_0^+$ denotes continuous time and $\dot{\boldsymbol{x}}(t) = \frac{d\boldsymbol{x}}{dt}$ is the change of $\boldsymbol{x}$ w.r.t. to time evaluated at time t.

Furthermore, a control system may have a derived output that is a linear combination of its states and inputs, which may restrict the observability of the state space from the output space (think of a boiler in which the internal pressure has a dynamic effect on the temperature, but cannot be measured from the outside because only a

temperature sensor was installed).

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t). \tag{2.8}$$

Discretisation of a continuous dynamical system model turns the ODE into a difference equation, assuming zero-order hold for the input $\boldsymbol{u}$ (i.e. the value $u(t)$ is fixed for the integration period), into

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_d\boldsymbol{x}_k + \boldsymbol{B}_d\boldsymbol{u}_k, \tag{2.9}$$

$$y_k = \boldsymbol{C}_d\boldsymbol{x}_k + \boldsymbol{D}_d\boldsymbol{u}_k, \tag{2.10}$$

where

$$\boldsymbol{A}_d = e^{\boldsymbol{A}T_s}, \tag{2.11}$$

$$\boldsymbol{B}_d = \int_{t=0}^{T_s} e^{\boldsymbol{A}t}dt\ \boldsymbol{B} = \boldsymbol{A}^{-1}(\boldsymbol{A}_d - \boldsymbol{I})\boldsymbol{B}, \tag{2.12}$$

$$\boldsymbol{C}_d = \boldsymbol{C}, \tag{2.13}$$

$$\boldsymbol{D}_d = \boldsymbol{D}, \tag{2.14}$$

and T_s is the sample time. Then

$$\forall k \in N_0, \boldsymbol{x}(kT_s) = \boldsymbol{x}_k \text{ and } \boldsymbol{y}(kT_s) = \boldsymbol{y}_k.$$

The eigenvalues of $\boldsymbol{A}_d$ and those of $\boldsymbol{A}$ are related as follows: $\hat{\lambda}_i = e^{\lambda_i T_s}$ where $\hat{\lambda}_i \in \sigma(\boldsymbol{A}_d)$ and $\lambda_i \in \sigma(\boldsymbol{A})$, where $\sigma(\boldsymbol{A})$ is the spectrum of matrix $\boldsymbol{A}$.

Remark 1. *The witnessed maximum amplitude of the discrete signals $\boldsymbol{x}_k, \boldsymbol{y}_k$ may be smaller to that of $\boldsymbol{x}(t), \boldsymbol{y}(t)$ due to synchronism at fraction-frequency sampling. This means that safety verification of the state space must account for this condition.*

Imagine that a signal $\boldsymbol{x}(t) = sin\left(\omega_f t\right)$. If we sample this signal at intervals $T_s = \frac{\pi}{\omega_f}$ starting at $t = 0$, then all measured samples will be 0. For this reason, systems must be sampled at intervals smaller than the maximum composing frequency of the source signal $T_s < \frac{\pi}{\omega_{max}}$. This is the so called Nyquist Criterion [20]. When $T_s < \frac{\pi}{2\omega_{max}}$, the maximum amplitude of the original signal can be upper bounded from the observed values using the cosine of the sampling function as follows:

$$\forall t \in [T_1,\, T_2] \text{ where } T_2 = T_1 + T_s,\, \max\left(\boldsymbol{x}(t)\right) < \frac{\max\left(\boldsymbol{x}(T_1), \boldsymbol{x}(T_2)\right)}{cos(T_s \omega_{max})}.$$

Since Equation (2.9) models (2.7), we may use the semantics defined in Section 2.2.3 to analyse continuous dynamical systems.

2.2.6 Model checking

We have defined the semantics of the model of our target systems, and the properties we are interested in verifying. Now we need to understand how we can use these paradigms to help us verify real systems. For this purpose we will use model checking [53, 54]. As its name suggests, model checking verifies a model against a specification using the model that represents it. Not all properties are necessarily verified using the same model. Instead, different properties may use different models and abstractions that preserve said properties. In subsequent sections we will discuss further modelling paradigms that will help us verify additional properties. For the moment we will discuss the case of safety, which is often expressed as a reachability problem.

Due to the intractability of some models, many tools use bounded model checking, which corresponds to the verification of the model in a bounded time horizon [28]. As an example we can see that taking each of the initial states and performing the operations given by Equation (2.1) for a number of steps would yield the sets of states visited at each iteration. Verifying that each set of state remains within the safe region is a set inclusion which corresponds to bounded model checking up to the given number of steps. This approach is costly and imprecise (since it relies on a discretisation of continuous

states), which is why we use abstraction (i.e. a model of the model) to obtain a sound and fast approach that can be verified over an infinite number of steps. In other words, we aim to perform bounded and/or unbounded model checking over a finite state space that over-approximates a continuous space through abstract representations.

For the sake of clarity we will point out that our models are intended for the verification of reachable states, and we will not examine deadlocks (when the model is prevented to progress in time) or Zeno behaviour (when the model executes an infinite number of discrete transitions in a limited amount of time).

2.3 Numeric computation and rounding errors

2.3.1 Numeric representations and soundness

In embedded software programs interacting and/or simulating physical plants it is of great importance to understand the effects of the chosen numeric representation for the continuous state space. These representations are almost invariably approximations to the original, which are assumed to be precise enough as to not cause any trouble, but it is precisely this assumption that often results in the most difficult and dangerous errors. There are generally speaking two paradigms for representing numbers in programs: fixed and variable precision numbers. We refer to both of these as finite word length representations (FWL). They subdivide into a variety of data types such as integer bit-vectors, fixed or floating point numbers and rationals. In this section we will discuss fixed and floating point numbers.

On one hand, we have variable word length types, which will use as much memory as needed to precisely represent the original domain (usually the reals). In this case, the limitations are caused by memory and speed, both of which are directly affected by the increasingly larger word lengths. Multiple Precision Floating Point Arithmetic and Software Rationals are good examples of these cases. It is possible for some of these data types to have memory constraints, at which point they fall into the second category. Fixed precision data types are most common in software programs. Whether

they be Floating Point or Fixed Point (of which integers are the special case for 0 decimal places) numbers, the problem arises in the misrepresentation of the reals (or equivalent data type being represented). Let $\mathbb{R}_{\langle M,F\rangle}$ (with $M \in N$ and $F \in \mathbb{Z}$) be a fixed point domain with M bits representing the binary word length of the representation and F bits representing the pre-scaling 2^{-F} (i.e., the number of bits after the decimal place). We note that a negative F denotes an integer multiplication that adds $-F$ trailing zeros in the number represented by M.

Let also $\mathbb{R}_{\langle M,[E]\rangle}$ be a floating point domain with M bits mantissa and E bits exponent. This domain corresponds to

$$\mathbb{R}_{\langle M,[E]\rangle} = \bigcup_{F} \mathbb{R}_{\langle M,F\rangle} \text{ where } F \in [-2^{E-1}, \ldots, 2^{E-1}].$$

This definition is important because it means that the distance between any two numbers in the domain depends on the location of those numbers. Since we have shown that a floating point domain is a union of a number of fixed point domains, we will continue our reasoning on fixed point only, which will apply to the floating point domain by extension. Operations on these numbers will incur errors that can be formaly evaluated and boud [36]. We examine the most relevant ones (and corresponding upper bounds used in this work):

1. Truncation

   Let $\mathcal{F}_{\langle M,F\rangle} : \mathbb{R} \rightarrow \mathbb{R}_{\langle M,F\rangle}$ be a truncation function, defined as

   $$x_{\langle M,F\rangle} = \mathcal{F}_{\langle M,F\rangle}(x) = x - \delta \text{ where } \delta = mod_F(x, x_{\langle M,F\rangle}),$$

   where $mod_F(\cdot)$ is the modulus operation performed on the last bit of the representation (2^{-F}).

   The number δ corresponds to the truncation error of the representation and it will propagate across operations.

2. Rounding

   Every time an operation is performed on a number in the $\mathbb{R}_{\langle M,F\rangle}$ domain, an error

may be introduced due to truncation of the result. The following errors appear in simple operations:

(a) Addition/Subtraction

$$x_{\langle M,F_1\rangle} \pm y_{\langle M,F_2\rangle} = z_{\langle M,F\rangle} + \delta, F = \min(F_1, F_2),$$

$$\text{where } |\delta| \leq 2^{-F} \text{ and } F_1 = F_2 \Rightarrow \delta = 0.$$

(b) Multiplication

$$x_{\langle M,F_1\rangle} * y_{\langle M,F_2\rangle} = z_{\langle M,F\rangle} + \delta, \text{ where } F = F_1 + F_2 \text{ and } |\delta| \leq 2^{-F}.$$

We note here that not all systems truncate equally. In the definition above, δ may be positive or negative, and this decision can be made dependent on the sign of x. Common cases in computer systems are: round downwards (i.e., to $-\infty$), round upwards (i.e., to $+\infty$), and round to zero. Rounding errors are cumulative which means the overall error will statistically increase with every operation, thus single transition algorithms can be more precise in this respect than iterative ones because they perform fewer operations needing rounding.

3. Overflow

Another source of error introduced by FWL representations is that of overflow. The finite word length set a maximum representable number in the domain $\pm\overline{M}$ where $\overline{M} = 2^{M-F} - 2^{-F}$, which means that any number outside this range cannot be accurately represented by the domain (*eg* $\delta = x - \overline{M}$). In the case of floating point, a special cases for infinity has been introduced, which addresses some of these issues, but once the error has gone to infinity, the numbers become meaningless. It is the responsibility of software designers and tools to account for overflows and either supply saturation functions to prevent them or raise alarms in their presence. In the case of verification, the latter is seen as a safety problem.

4. Zeno behaviour

   While this work does not explore Zeno behaviours, it is worth noting that this effect may be caused by rounding errors in floating point programs. The reason for this is that if an operation contains two numbers which belong to disjoint sub-domains (*ie* neither number can be represented in the domain of its counterpart), the resulting operation may become a no-op/skip, subsequently destabilizing the program by creating infinite loops.

We will use the domain $\mathbb{R}_{\langle C \rangle}$ where $\langle C \rangle = \langle M_c, F_c \rangle$ to represent a digital implementation of LTI dynamics which we will refer to as the controller, and $\mathbb{R}_{\langle P \rangle} \supseteq \mathbb{R}_{\langle C \rangle}$ to represent the domain of a digital model of the dynamics of a real plant when evaluated by a verification tool. Furthermore, we will refer to the errors incurred by the calculations performed in these domains as Finite Word Length (FWL) effects.

In order to overcome the limitations of the FWL representation, this work explores the use of interval arithmetics to retain soundness.

2.3.2 Interval arithmetics

Definition 1. *An interval domain is a domain over the reals that defines a set of numbers by an upper and lower bound. We use the notation*

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}, \tag{2.15}$$

$$]x[=]\underline{x}, \overline{x}[= \{x \in \mathbb{R} \mid \underline{x} < x < \overline{x}\}, \tag{2.16}$$

$$[x[= [\underline{x}, \overline{x}[= \{x \in \mathbb{R} \mid \underline{x} \leq x < \overline{x}\}, \tag{2.17}$$

to represent an element of the domain. We also define the operators $+, -, *$[1], $/$ *as the sum, subtraction, multiplication and division of an interval, with the following properties:*

$$[x_1] + [x_2] = [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2], \tag{2.18}$$

$$[x_1] - [x_2] = [\underline{x}_1 - \overline{x}_2, \overline{x}_1 - \underline{x}_2], \tag{2.19}$$

$$[x_1] * [x_2] = [\inf(\underline{x}_1\underline{x}_2, \underline{x}_1\overline{x}_2, \overline{x}_1\underline{x}_2, \overline{x}_1\overline{x}_2), \sup(\underline{x}_1\underline{x}_2, \underline{x}_1\overline{x}_2, \overline{x}_1\underline{x}_2, \overline{x}_1\overline{x}_2)], \tag{2.20}$$

$$\frac{[x_1]}{[x_2]} = \left[\inf\left(\frac{\underline{x}_1}{\underline{x}_2}, \frac{\underline{x}_1}{\overline{x}_2}, \frac{\overline{x}_1}{\underline{x}_2}, \frac{\overline{x}_1}{\overline{x}_2}\right), \sup\left(\frac{\underline{x}_1}{\underline{x}_2}, \frac{\underline{x}_1}{\overline{x}_2}, \frac{\overline{x}_1}{\underline{x}_2}, \frac{\overline{x}_1}{\overline{x}_2}\right)\right] \textit{ given } 0 \notin [\underline{x}_2, \overline{x}_2]. \tag{2.21}$$

Note that division is not defined for denominator intervals containing 0. This case never applies to our algorithms, since pivoting, which is the only operation requiring division, operates on non-zero values. Additionally, any monotonic function over the reals can be mapped into an interval equivalent as:

$$f(x) \to [f]([x]) = [\inf(f(\underline{x}), f(\overline{x})), \sup(f(\underline{x}), f(\overline{x}))]. \tag{2.22}$$

In the case of non-monotonic functions, such as trigonometric operations, the calculation of $[f]([x])$ has to take into consideration the minima and maxima of the function over the interval $[x]$.

The above definitions are over the reals, which means that the result of any of these operations is exact. However, when using FWL representations such as floating points, results may not be representable in the given format and are implicitly rounded. In order for interval arithmetics to be sound, these domains must always round the upper limit upwards and the lower limit downwards, thus expanding the interval. These operations are well defined within the IEEE-754 standard [104] as well as in existing multiple precision libraries, such as mpfr [78], so we will not discuss the details here. For a full description of interval arithmetics over IEEE-754 floating point representations see [141].

[1] As standard, we will often omit this operator in the following.

2.3.3 Numerical eigendecomposition

In many cases, transforming the problem into some canonical form reduces its complexity. One of the most common canonical forms is the Jordan Canonical Form [162] which has been used in Section 2.5.2. The process of extracting the Jordan Form of a matrix is called eigendecomposition. Broadly, there are two modern approaches to eigendecomposition: symbolic and numeric. Symbolic approaches are exact and sound, but they are slow and do not scale well. Computationally, they rarely deal with matrices larger than 10x10, whereas numeric approaches can manage square matrices with thousands or tens of thousands of rows. Two common approaches are used in modern tools, power series and QR decomposition [162]. In this work we focus on the latter. The QR decomposition approach extracts the eigenstructure of a matrix by first finding an upper diagonal equivalent.

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \text{ where } \boldsymbol{Q}^{\intercal} = \boldsymbol{Q}^{-1}.$$

The orthogonality of $\boldsymbol{Q}$ is exploited to iteratively find the eigenvalues as follows: let $\boldsymbol{A}_0 = \boldsymbol{Q}_0\boldsymbol{R}_0 = \boldsymbol{A}$. While $\boldsymbol{A}_i$ is not diagonal, we do $\boldsymbol{A}_{i+1} = \boldsymbol{R}_i\boldsymbol{Q}_i$. Note that $\boldsymbol{A}_{i+1} = \boldsymbol{R}_i\boldsymbol{Q}_i = \boldsymbol{Q}_i^{\intercal}\boldsymbol{Q}_i\boldsymbol{R}_i\boldsymbol{Q}_i = \boldsymbol{Q}_i^{\intercal}\boldsymbol{A}_i\boldsymbol{Q}_i$, has therefore the same eigenvalues as $\boldsymbol{A}$. Given n iterations, and $\boldsymbol{A}_n$ diagonal, the elements in said diagonal are the eigenvalues of $\boldsymbol{A}$, and since $\boldsymbol{A}_n$ is upper diagonal, we can calculate its eigenvectors one element at a time (then multiply by $\boldsymbol{Q} = \prod_i \boldsymbol{Q}_i$ to get the eigenvectors of $\boldsymbol{A}$).

Although this algorithm is fast and efficient, it relies on the convergence towards zero of the entries of the lower diagonal of $\boldsymbol{R}_i$, which will take decreasing values around zero (typically alternating the sign while decreasing the magnitude). This means however, that while in most cases the algorithm is known to converge rapidly, this is not generally the case when using interval arithmetics. In fact, whilst the centre of the interval converges to 0, the upper and lower limits will commonly diverge to $\pm\infty$. For this reason, numerical eigendecomposition of the matrices must be performed using non-interval arithmetics and the error bounds determining the range of the interval found a-posteriori. Operations on finding these bounds are discussed in Section 6.1.2.

2.4 Abstract domains

Model checking as previously described has many limitations. Among these, the state space explosion problem is probably the most significant one. Since we are dealing with an uncountable set of states (given that our problem domain is in the reals), we need to enable our technique to reason about all possible states without evaluating each of them individually. For this we use abstractions, which allow us to reason about sets of states rather than individual ones. For example, interval arithmetics are an abstraction of standard arithmetics.

2.4.1 Abstract interpretation

In cases where models are too rich in information for efficient evaluation, it is possible to reduce the amount of information by summarizing the data in a way that retains the properties we are interested in. Abstract Interpretation [57] is a method that creates a sound approximation of a domain with respect to the desired properties by creating a lattice that preserves these properties when mapping it into the original domain. Formally, given a concrete domain C and an abstract domain $\mathcal{A}$, we define an abstract function

$$\alpha : C \to \mathcal{A},$$

where $\alpha(X)$ is the called abstraction of X. As and example, the set of integers $X = \{1, 2, 4\}$ may be abstracted into $\alpha(X) = [1, ..., 4]$. The advantage of this mapping is that given an appropriate abstract domain, for some relation $f : C \to C$, there exists $f' : \mathcal{A} \to \mathcal{A}$, which is expected to be computationally less expensive, although imprecise. In our example, the addition of the interval to itself is obtained by performing two additions ($\alpha(X) + \alpha(X) = [1 + 1, ..., 4 + 4] = [2, ..., 8]$), whereas the concrete case requires three ($X + X = \{1 + 1, 2 + 2, 4 + 4\} = \{2, 4, 8\}$). However, since we are interested in $f(X)$, we

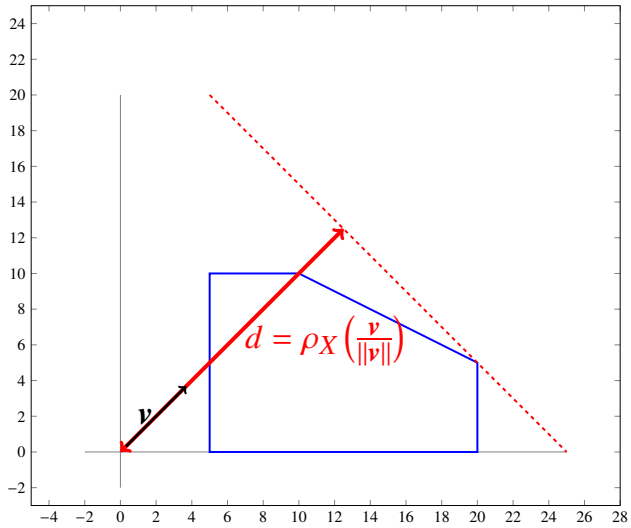

Figure 2.1: Support function for a polyhedral set in $\mathbb{R}^2$. The distance between the tangent line and the origin is $\rho_X\left(\frac{v}{\|v\|}\right)$.

also need a concretisation function that preserves soundness:

$$\gamma : \mathcal{A} \to C, \text{ such that } X \subseteq \gamma(\alpha(X)) \text{ and } f(X) \subseteq \gamma\left(f'(\alpha(X))\right), \tag{2.23}$$

where the ideal abstractions are those in which $X = \gamma(\alpha(X))$. Returning to our example, $\gamma(\alpha(X)) = \{1, 2, 3, 4\} \supset \{1, 2, 4\}$. The implicit relation between α and γ is a Galois connection [58]. Notice that in abstractions where $X \subset \gamma(\alpha(X))$, recursive calls of $\gamma(\alpha(\cdot))$ may result in over-approximations that may eventually lead to top. Another important point relevant to this work is that the sets X and $\gamma(\alpha(X))$ need not be finite even when $\alpha(X)$ is, as for example in the case of real arithmetic intervals. This means that we can map a concrete continuous space into a discrete abstract one, which is paramount to the abstractions used in this work. Support functions 2.4.2, convex polyhedra 2.4.3, interval analysis 2.3.2, and abstract acceleration 2.5 are all cases of abstract interpretation used in this work.

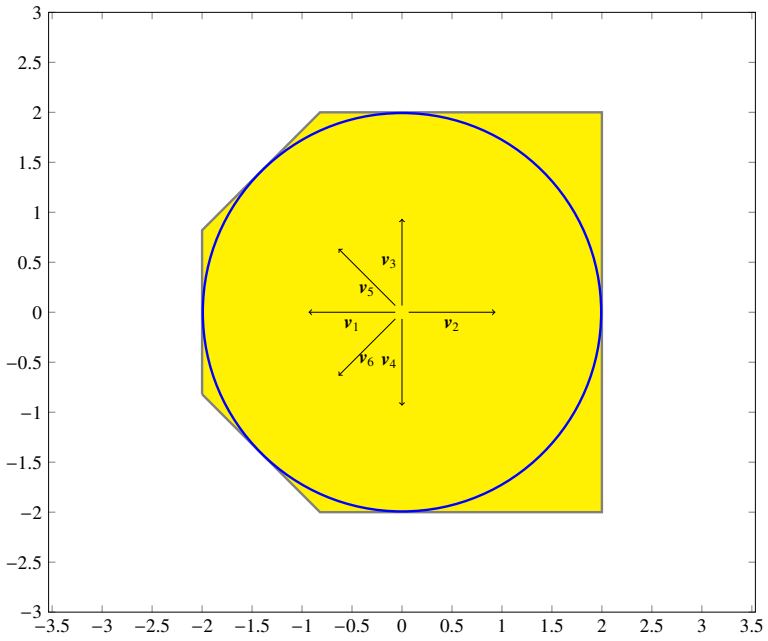

Figure 2.2: Support function for a circular set using six directions ($\boldsymbol{v}_1 \cdots \boldsymbol{v}_6$). The resulting polyhedron is an over-approximation of the original set (note that directions need not be symmetrical).

2.4.2 Support functions

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 2.1.

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 centred 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 obtained by sampling the support function in a number of directions may generate a polyhedron (see Figure 2.2), which we will discuss in the next section. Finitely sampled support functions (i.e. using a limited number of directions) are template polyhedra (see 2.4.6) in which the directions are not fixed,

which helps avoiding wrapping effects [92] (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

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

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 $X \subseteq \mathbb{R}^p$, represented by a general assertion $\theta(X)$, 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

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

Properties of support functions

Several properties of support functions allow us to reduce the complexity of operations. The most significant are [90]:

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

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.

Support functions in complex spaces

The literature does not state, to the best of our knowledge, any description of the 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 their operation on complex spaces.

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

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

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

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

The imaginary part of the result of the dot product is not relevant to the support function, therefore we ignore it.

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

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

Using abstract interpretation we may 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 may therefore establish an equivalency $\rho_X(\boldsymbol{v}) = \rho_{X'}(\boldsymbol{v}')$. □

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 using it as a multiplier 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 pseudoeigenvalue representation (See Equation (2.6)). 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 problem statement, 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.

2.4.3 Convex polyhedra

A polyhedron is a subset of $\mathbb{R}^p$ with hyperplanar 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 the interior of the polyhedron. 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 [86]. The mechanism to switch between these two representations is given in Section 2.4.4.

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 is a set $V \in \mathbb{R}^p$ such that $\boldsymbol{v} \in V$ is a vertex of the polyhedron.

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 the intersection of p or more), and is described mathematically as a function of the vector that is normal to the hyperplane (note that this representation is made minimal when 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

$$X = \{\boldsymbol{x} \in \mathbb{R}^p \mid \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d}, \boldsymbol{C} \in \mathbb{R}^{m \times p}, \boldsymbol{d} \in \mathbb{R}^m\}, \tag{2.24}$$

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:

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

2.4.4 Operations on convex polyhedra

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

Translations

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

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

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

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

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

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

where $\boldsymbol{L}^{+}$ represents the pseudo-inverse of $\boldsymbol{L}$ [137]. In the case when the inverse $\boldsymbol{L}^{-1}$ exists, then

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

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.

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

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

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 $X_1 = \{\boldsymbol{x} \mid \boldsymbol{C}_1\boldsymbol{x} \leq \boldsymbol{d}_1\}, \quad X_2 = \{\boldsymbol{x} \mid \boldsymbol{C}_2\boldsymbol{x} \leq \boldsymbol{d}_2\}, \quad$ be two sets, then

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

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

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

Set Hadamard products

Definition 2. *Given two vertex representations V' and V'', we define the set Hadamard product operation using vertex representations as:*

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

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

Lemma 1. *The set defined by $V = V' \circ V''$ is a convex set containing all possible combinations of products between elements of each set.*

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

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

which extends to multiple points [41]

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

If we apply the Hadamard product we get

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

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 $\sum_{ij=1}^{|V'||V''|} k_{ij} = \sum_{i=1}^{|V'|} \sum_{j=1}^{|V''|} k_i k_j = 1$.

This proves the theorem. □

We note that in the case of 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

$$X \subseteq \{\boldsymbol{x} | \boldsymbol{x} \cdot \boldsymbol{t} < \max(\{\rho_{X_2}(\boldsymbol{t} \circ \boldsymbol{v}') \mid \boldsymbol{v}' \in V'\}), \boldsymbol{t} \in T\}, \tag{2.25}$$

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_1.

Vertex enumeration

The vertex enumeration problem corresponds to the algorithm required to obtain 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 $\boldsymbol{V}\boldsymbol{x} \leq [\ 1 \quad 1 \quad \cdots \quad 1\ ]^\intercal$ with $\boldsymbol{V} = [\ \boldsymbol{v}_1 \quad \cdots \quad \boldsymbol{v}_m\ ]^\intercal, \boldsymbol{v}_i \in V$. 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 [21] is a reverse search algorithm, while cdd [86] 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

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

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

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

where V is called the generator of X and C the cone 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 (hence the extreme rays of C) 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 2.4.4).

We also point out that

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

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 $\boldsymbol{C}_K \in \mathbb{R}^{n_K \times p}$, $\boldsymbol{C}_{i,*} \in \mathbb{R}^{1 \times p}$, $V_K \in \mathbb{R}^p$,

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

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 [86],

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

2.4.5 Linear optimisation

There is one operation on convex polyhedra that deserves special attention. Linear optimisation is the problem of finding the maximum value of a linear objective function given a set of linear constrains. This is equivalent to finding the support function of a convex polyhedron in a given direction and it is key to the development of our technique. One of the most popular algorithms for solving linear optimisation problems is the simplex algorithm, on which we will build upon in the course of this work.

The revised simplex method

The Simplex method [61] is an algorithm devised to solve linear program optimisation problems. It is a so-called exterior point method because it traverses a polyhedron along its edges in order to find the support function of a polyhedron in a given direction (which will always be found on one of the vertices). Formally we seek to maximise $\boldsymbol{v}^\intercal \boldsymbol{x} \mid \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d}$ where $\boldsymbol{v}$ is the desired direction, and the representation on the right corresponds to a

convex polyhedron in $\mathbb{R}^n$. The simplex algorithm solves this problem by visiting adjacent vertices on the polyhedra, moving in the general direction of the objective vector $\boldsymbol{v}$. In order to achieve this, the simplex algorithm first introduces a number of slack variables to convert the system into equalities, accounting for the difference in each of the inequalities (*ie* for each row i in matrix $\boldsymbol{C}$, we add a value $\boldsymbol{x}_{si}$ such that $\boldsymbol{C}_{i,*}\boldsymbol{x}+\boldsymbol{x}_{si} = \boldsymbol{d}_i \Rightarrow \boldsymbol{C}\boldsymbol{x}+\boldsymbol{x}_s = \boldsymbol{d}$). The solution is feasible as long as all the slack variables are non-negative. The resulting system is described by the formula

$$\begin{bmatrix} 1 & -\boldsymbol{v} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} h_a \\ \boldsymbol{x} \\ \boldsymbol{x}_s \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{d} \end{bmatrix}.$$

where $h_a = \rho_{\boldsymbol{x}}(\boldsymbol{v}) = \boldsymbol{x}\cdot\boldsymbol{v}$ is the value of the support function of the point $\boldsymbol{x}$ in the direction $\boldsymbol{v}$.

The simplex method performs pivot operations on this matrix system, thus assigning new values to $[h_a\ \boldsymbol{x}\ \boldsymbol{x}_s]^\intercal$ in order to find the optimal solution that maximises h_a. A pivot operation is a row operation on a matrix such that one element in the matrix is normalised to 1 (by dividing the corresponding row by the element's value), and all elements in the corresponding column are brought to zero (by subtracting the elements row times the coefficient on the pivot column at all other rows). These pivots are performed in two stages. First, a feasible solution is sought by pivoting on the rows that do not match the inequality, and then a maximum is found through maximising the value of each element in the objective row by pivoting on a row that effects a positive change in said variable.

Although it has an exponential worst case complexity, the simplex algorithm typically performs faster than polynomial time algorithms, making it very efficient for linear decision procedures. However, the matrix system described above can grow very large and is hard to adapt to accumulated errors. For this reason, we look at the revised simplex algorithm [163] which reformulates the problem to a minimal expression.

The equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{A} = [\ \boldsymbol{0}\quad \boldsymbol{C}\quad \boldsymbol{I}\ ] \in \mathbb{R}^{m\times p+m}$ has a basic solution $\boldsymbol{x} \in \mathbb{R}^{p+m}$ if $\boldsymbol{x}$ can be partitioned into p *basic* variables such that the columns of $\boldsymbol{A}$ corresponding to these variables form a non-singular basis, and m *non-basic* variables whose value is 0.

Initially, the elements of $\boldsymbol{x}_s$ in the simplex algorithm correspond to the *non-basic* variables. Pivot operations will swap a *basic* variable with a *non-basic* one, thus changing the basis.

Let $\boldsymbol{B}$ be the basis found by selecting the columns of $\boldsymbol{A}$ that correspond to the *basic* variables. Since the effect of a row operation can be represented by multiplying a matrix from the left, the effect of all row operations leading to the columns of $\boldsymbol{A}$ corresponding to the new set of *basic* variables to become the identity is equivalent to multiplying by the inverse of $\boldsymbol{B}$. Thus

$$\boldsymbol{B}^{-1}\begin{bmatrix} \mathbf{0} & \boldsymbol{C} & \boldsymbol{I} \end{bmatrix}\begin{bmatrix} h_a \\ \boldsymbol{x} \\ \boldsymbol{x}_s \end{bmatrix} = \boldsymbol{B}^{-1}\boldsymbol{d}. \tag{2.26}$$

Partitioning $\boldsymbol{x}_B, \boldsymbol{v}_B, \boldsymbol{d}_B$ as the rows of $\boldsymbol{x}$, $\boldsymbol{v}$ and $\boldsymbol{d}$ corresponding to the basic variables, and $\boldsymbol{B}_B^{-1}$ the corresponding columns of $\boldsymbol{B}^{-1}$, the above equations simplify to:

$$\begin{bmatrix} h_a \\ \boldsymbol{x}_B \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{v}_B\boldsymbol{B}_B^{-1} \\ 0 & \boldsymbol{B}_B^{-1} \end{bmatrix}\begin{bmatrix} 0 \\ \boldsymbol{d}_B \end{bmatrix}. \tag{2.27}$$

The revised simplex algorithm works by updating the matrix $\boldsymbol{B}^{-1}$ rather than working directly on $\boldsymbol{C}$ Some additional advantages of this approach, exploited in this work, are discussed in Section 6.1.3.

The dual simplex algorithm and generalised simplex methods

The revised simplex method looks at a way of managing pivoting more efficiently. However, the performance of the simplex algorithm is largely dependent on the selection of the pivots. For example, the dual simplex method [23] approaches the problem from a complementary angle: the primary simplex algorithm looks to find an optimal solution whilst remaining feasible, whereas the dual simplex algorithm seeks to find a feasible solution while remaining optimal. This is achieved by pivoting first to an optimal basis that may not be feasible and performing subsequent pivots that seek to make the basis feasible. We make use of the dual simplex algorithm in this work because it is more

convenient for our implementation. In particular, given that we are interested in over-approximations, it is possible to stop the algorithm before finding a feasible solution as long as the current basis satisfies our properties.

Another improvement over the simplex algorithm is the generalised simplex method [61]. In this case the objective is to deal with degeneracy (which happens when one of the inequalities of the simplex does not correspond to a face of the enclosed polyhedra). The problem with degeneracy is that it allows the simplex to cycle between two bases without optimizing the solution. The generalised simplex solves this problem by partitioning the inequality set in order to ensure that pivots are always progressing the objective formula (i.e. making it more optimal in the case of the primary simplex). Other improvements such as Bland's rules [30] and pivoting decisions based on heuristics have also been developed, but are clearly outside the scope of this work. Due to time restrictions, and because this work focuses on abstract acceleration, we do not implement these latter improvements of the simplex, which may be added to our tool at a later time (this is indeed a place for improvement in our technique).

2.4.6 Template polyhedra

Template polyhedra seek to exploit the last property described in the previous section in order to optimise operations in the abstract domain. Namely, rather than representing polyhedra in an arbitrary set of directions, we define a set of template directions that are to be used for all polyhedra. This is equivalent to having $\boldsymbol{C_1} = \boldsymbol{C_2} \cdots = \boldsymbol{C_n}$ for all polyhedra in the domain. The most basic example of template polyhedra are hyper-boxes, where all but one of the columns in each row of $\boldsymbol{C}$ is allowed to have a non-zero value (typically 1 or -1). The improved efficiency is given by the properties described in the previous section, but the drawback is that the resulting polyhedron of a linear transformation will have to be 're-templated' to match the description. This operation creates an over-approximation of the polyhedron, also called the 'wrapping effect' since the new polyhedron is often a larger one containing the direct result. For example, rotating a hyper-box in this domain will result in a larger hyper-box containing the original (see figure 2.3).

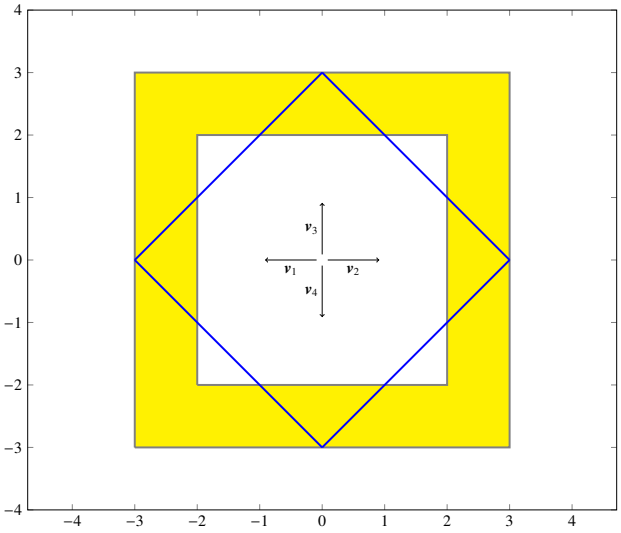

Figure 2.3: Wrapping effect on a rotating hyper-box using template polyhedra. White: original polyhedron. Blue: projection of the polyhedron caused by the transformation. Yellow: result of the transformation using template polyhedra in the direction of the axes ($\boldsymbol{v}_i$).

The logahedral domain

Since one of the contributions of this work is the use of support functions to replace the logahedral domains used in existing work, we briefly describe their nature. The logahedral domain is a specialisation of template polyhedra with two characteristics:

- The directions of the polyhedra are restricted to two dimensional spaces.
- The non-zero elements are integer powers of 2.

This may be expressed mathematically as follows:

$$a_i \boldsymbol{x}_i + a_j \boldsymbol{x}_j \leq k, \text{ with } a_i, a_j \in \{-2^n, 0, 2^n \mid n \in \mathbb{Z}\}, i \neq j, \text{ and } k \in \mathbb{R}$$

The order of a logahedral domain is the absolute maximum value of any $\boldsymbol{x}_j$ in the templates of the polyhedra.

The logahedral domain is often used by researchers for its speed since the integer values are fast to process and the templated polyhedra are easier to manipulate.

2.5 Abstract matrices

We have defined a framework to reason about sets which allows us to deal with continuous spaces without enumerating them. We now need to do the same for continuous time. Abstract acceleration does this by reasoning about the combined dynamics that define a set of transitions. This requires two stages: acceleration and abstraction. The resulting abstraction is a matrix that represents all possible transitions up to a given time horizon. This abstraction is what we call an abstract matrix domain. The simplex form of an abstract matrix is an interval matrix, in which each element corresponds to an interval that contains any possible valuation of the dynamics for that position in the matrix. Imagine a 2-dimensional model with dynamics

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \boldsymbol{x}_k = 0.5\boldsymbol{x}_k.$$

The abstract matrix representing the dynamics of this model for an infinite time horizon is the interval matrix in

$$\hat{\boldsymbol{x}}^{\#}_{k+1} = \begin{bmatrix} [0, 0.5] & 0 \\ 0 & [0, 0.5] \end{bmatrix} \boldsymbol{x}_k = [0, 0.5]\boldsymbol{x}_k.$$

We will see next how to compute such matrices in the general case.

2.5.1 Acceleration techniques

Acceleration of a transition relation is a method that seeks to precisely describe the transition relations over a number of steps using a concise description of the overall transition between the first and final step. Namely, it looks for a direct formula to calculate the post-image of a loop from the initial states of the loop. It may be applied to both continuous and discrete time models using different paradigms. Formally, given the dynamics in Equation (2.1), an acceleration formula aims at computing the reach set (2.3) using a function f such that $f(\cdot) = \tau^n(\cdot)$. Let us consider the transition relation $\tau(x) = x+1$.

Computing $\tau^3(x)$ results in $x + 1 + 1 + 1 = x + 3$. The right hand side of this equation corresponds to the acceleration of $\tau^3(x)$, which can easily be applied to $\tau^n(x) = x+n$. In the case of discrete time LTI models without inputs, the corresponding equation is $\boldsymbol{x}_n = \boldsymbol{A}^n\boldsymbol{x}_0$. We will use this property and others derived from it to calculate our abstract matrices.

2.5.2 Computation of abstract matrices

Definition 3. *An abstract matrix $\mathcal{A}^n \in \mathbb{R}^{p\times p}$ is an over-approximation of the union of the powers of the matrix $\boldsymbol{A}^k$, such that $\mathcal{A}^n \supseteq \bigcup_{k\in[0,\dots,n]} \boldsymbol{A}^k$. Its application to the initial set X_0 results in*

$$\hat{X}_n^\sharp = \mathcal{A}^n X_0, \tag{2.28}$$

such that $\hat{X}_n^\sharp \supseteq \hat{X}_n$ is an over-approximation of the reach tube described in Equation (2.4)*. Since this is a multiplication of two polyhedral sets, equation* (2.28) *is evaluated using equation* (2.25)*.*

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 2.5.3. Similar to [108], 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 generalised 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

$$\boldsymbol{J}^n = \begin{bmatrix} \boldsymbol{J}_1^n & & \\ & \ddots & \\ & & \boldsymbol{J}_r^n \end{bmatrix}, \quad \text{where} \quad \boldsymbol{J}_s^n = \begin{bmatrix} \lambda_s^n & \binom{n}{1}\lambda_s^{n-1} & \dots & \binom{n}{p_s-1}\lambda_s^{n-p_s+1} \\ & \lambda_s^n & \binom{n}{1}\lambda_s^{n-1} & \vdots \\ \vdots & & \ddots & \vdots \\ & & & \lambda_s^n \end{bmatrix}. \tag{2.29}$$

The abstract matrix $\mathcal{A}^n$ is computed as an abstraction over a set of vectors $\boldsymbol{m}^k \in \mathbb{R}^p, k \in [1, \ldots, n]$ of distinct entries of $\boldsymbol{J}^k$, as explained below.

Let $\boldsymbol{I}_s = [\ 1 \quad 0 \quad \cdots \quad 0\ ] \in \mathbb{R}^{p_s}$. The vector $\boldsymbol{m}^k$ is obtained by the transformation φ^{-1} (which is always invertible) as

$$\boldsymbol{m}^k = \varphi^{-1}\left(\boldsymbol{J}^k\right) = \left[\ \boldsymbol{I}_1\boldsymbol{J}_1^k \quad \cdots \quad \boldsymbol{I}_r\boldsymbol{J}_r^k\ \right]^\intercal \in \mathbb{R}^p, \tag{2.30}$$

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

If $\boldsymbol{J}$ is diagonal [108], then $\boldsymbol{m}^k$ results in the vector of powers of eigenvalues $[\ \lambda_1^k \quad \cdots \quad \lambda_p^k\ ]$. The abstract matrix is thus bound by the intervals

$$\begin{cases} \left[\min\{|\lambda_s|^0, |\lambda_s|^n\}, \max\{|\lambda_s|^0, |\lambda_s|^n\}\right] & \lambda_s \in \mathbb{R}^+ \\ \left[-\max\{|\lambda_s|^0, |\lambda_s|^n\}, \max\{|\lambda_s|^0, |\lambda_s|^n\}\right] & otherwise \end{cases}, s \in [1, \ldots, r] \text{ where } r = p. \tag{2.31}$$

We observe that the spectrum of the abstract matrix $\sigma(\mathcal{A}^n)$ (exemplified in Equation (2.31)) over-approximates $\bigcup_{k\in[1,\ldots,n]} \sigma(\boldsymbol{A}^k)$.

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 $\left(\lambda_s^n, \binom{n}{1}\lambda_s^{n-1}, \ldots, \binom{n}{p_s-1}\lambda_s^{n-p_s+1}\right)^\intercal$ of $\boldsymbol{J}_s^n$.

Since φ transforms the vector $\boldsymbol{m}$ into the shape of (2.29) of $\boldsymbol{J}^n$, it is called a *matrix shape* [108]. We then define the abstract matrix as

$$\mathcal{A}^n = \{\boldsymbol{S}\ \varphi(\boldsymbol{m})\ \boldsymbol{S}^{-1} \mid \boldsymbol{\Phi}\boldsymbol{m} \leq \boldsymbol{f}\}\ , \tag{2.32}$$

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 distinct entries) within $\boldsymbol{J}$, we find an over-approximation

of the convex hull containing the points

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

The reason for evaluating the convex hull pairwise is twofold. In the first instance, we note that the set $\left\{\boldsymbol{m}^k \mid k \in [1, \dots, n]\right\}$ is not convex. This makes it hard to find its support in arbitrary directions. Ideal directions to choose, would be the normals to the gradients of the discrete funtion ∇m^k, which would provide the tightest over-approximation at iteration k, but as will be seen below, when combining negative or complex conjugate eigenvalues, the corresponding hyperplane tangent may cut the set in two and is therefore not part of its convex hull. The second reason for choosing pairwise directions is that we need an even distribution of the directions in $\mathbb{R}^p$, and it is easier to do this in a pairwise manner. Convex optimization techniques may be used to find a better set of directions, but they fall outside the scope of this work.

2.5.3 Abstract matrices in complex spaces

To deal with complex numbers in eigenvalues and eigenvectors, [108] 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 $[0, \overline{r}_s]$, where $\overline{r}_s = \max\{r_s^k \mid k \in [0, ..., n]\}$ (in the worst case scenario this is

over-approximated by a hyper-box with $\lambda_s^k \in [-\overline{r}_s, \overline{r}_s] + [-\overline{r}_s, \overline{r}_s]i$, but we will introduce tighter bounds in the course of this work).

2.6 Control theory

The semantics discussed in sections 2.2 do not describe all properties of interest in our models. In some cases, specific values or substructures define hidden properties which can only be inferred given an appropriate transformation. Since our models are being applied to linear control systems, and in particular to feedback systems with observers, we provide a summary of the important elements using control theory for their transformations.

A feedback system is one where the output is fed back into the input, thus making part or all of the system a closed system (i.e., no inputs). Such a system is exemplified in Figure 2.4. Since the feedback line creates a loop in the diagram this is known as a closed loop, and the overall behaviour of the resulting system is said to have closed loop dynamics. The feedback may be direct or have further transformations, some of which will be discussed in this section. For the moment we will also point out that in the models we are interested in, this feedback is digital and thus creates some further considerations regarding discretisation.

The objective of the feedback is to adhere to some properties we decide to control. In most cases these properties include stability. We define stability as the ability of the system output to remain bounded given a bounded input. This is known as BIBO stability. Further constraints to this property may be added to enforce stronger stability (e.g. convergent). We discuss the requirements we are interested in in the following.

2.6.1 Static output feedback and pole placement

Given an unstable plant with divergent dynamics, the problem of stabilizing the output is to find a controller that results in convergent closed loop dynamics. The term Static Output Feedback relates to the desired effect of having a static output given a fixed reference signal. In modern embedded systems, said controller is a digital program,

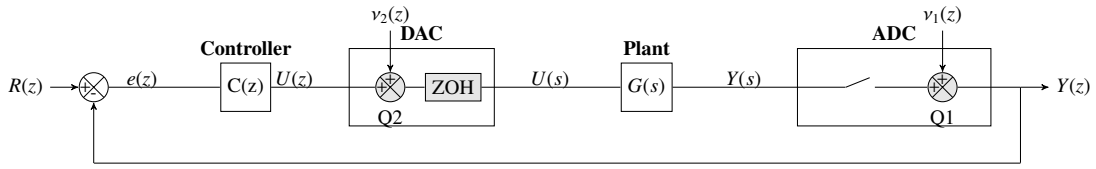

Figure 2.4: Closed loop digital control system

which means that the closed loop dynamics are subject to time and space discretisation as described in sections 2.2.5 and 2.3.1. In real a physical plant, these discretisations are performed by an Analog-to-Digital Converter(ADC) which samples the output of the plant at intervals T_s (time discretisation) and quantises it to an FWL representation (space discretisation). Conversely, a Digital-to-Analog Converter(DAC) transforms a discretised value into a real input to the physical plant. In general terms, when analysing such systems, the controller is known to have discrete dynamics, whereas the plant must be discretised based on the sample and hold characteristics. The ADC/DAC converters are considered to introduce a quantisation noise on the system as shown in Figure 2.4. Let the quantiser $Q1$ be the source of a non-deterministic noise ν_1 and $Q2$ be the source of a non-deterministic noise ν_2. The convergence of such a system is a frequency domain property, which is why we first analyse the system using its transfer function. The transfer function of a system relates its input and output in the frequency domain (more generally in the Laplace domain for continuous time systems and the Z-domain for discrete time). In the case of linear systems, these correspond to the characteristic polynomials of the system dynamics (i.e. the polynomial whose roots are the eigenvalues of the dynamics). The following equation models the system, encompassing the discretisation of the plant:

$$Y(z) = C(z)G(z)(R(z) - Y(z)) + \nu_1(z) + \nu_2(z)C(z)G(z), \tag{2.33}$$

The above can be rewritten as follows:

$$Y(z) = H_1(z)R(z) + H_2(z)\nu_1(z) + H_3(z)\nu_2(z), \tag{2.34}$$

where

$$H_1(z) = \frac{C(z)G(z)}{1 + C(z)G(z)}, \tag{2.35}$$

$$H_2(z) = \frac{1}{1 + C(z)G(z)}, \tag{2.36}$$

$$H_3(z) = \frac{G(z)}{1 + C(z)G(z)}. \tag{2.37}$$

Assumption 1. *The quantisation noises ν_1 (from $Q1$) and ν_2 (from $Q2$) are uncorrelated white noises and their amplitudes are always bounded by the half of quantisation step [19], i.e., $|\nu_1| \leq \frac{q_1}{2}$ and $|\nu_2| \leq \frac{q_2}{2}$, where q_1 and q_2, respectively, are the quantisation steps of ADC and DAC.*

A discrete-time linear system is said to be Bounded-Input and Bounded-Output (BIBO) stable if and only if every pole of its transfer function lies inside the unit circle [20]. Analyzing Eq. (2.34), the following proposition states necessary conditions for BIBO stability of the closed-loop system in Figure 2.4, since the exogenous signals $R(z)$, ν_1, and ν_2 are bounded. Hence, our close loop system is stable if the poles in $H_1(z), H_2(z)$ and $H_3(z)$ are inside the unit circle. Since all three transfer functions share the same poles, we only need to look at the roots of the polynomial

$$C_n(z)G_n(z) + C_d(z)G_d(z) : C(z) = \frac{C_n(z)}{C_d(z)} \wedge G(z) = \frac{G_n(z)}{G_d(z)}.$$

Any controller that ensures these roots lie within the unit circle is a stabilizing controller for the system.

This same analysis can be applied to MIMO systems where there is a transfer functions of the system for each combination of input-output. Rather than examine the transfer functions of these systems, we will first look at minimal descriptions of the state space dynamics that will help us extract them with little extra effort whilst providing us with tools to analyse other properties.

2.6.2 Controllable canonical form

Let us have a discrete time Single Input Single Output (SISO) system with a p^{th}, q^{th} order model with $p \geq q$.

$$y[k] = \sum_{i=1}^{p} a_{p-i} y[k-i] + \sum_{i=0}^{q-1} b_{p-i} u[k-i],$$

where y is the output of the system and u its input. The equivalent LTI dynamics of such a model are:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_c \boldsymbol{x}_k + \boldsymbol{B}_c u_k, \text{ where } \boldsymbol{x}_k = [\ y_k \quad y_{k-1} \quad \dots \quad y_{k-(p-1)}\ ]^\intercal, \tag{2.38}$$

$$y_k = \boldsymbol{C}_c \boldsymbol{x}_k + b_0 u_k,$$

$$\boldsymbol{A}_c = \begin{bmatrix} -a_{p-1} & -a_{p-2} & \cdots & -a_1 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}, \boldsymbol{B}_c = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix},$$

$$\boldsymbol{C}_c = [\ b_0 - a_0 b_p \quad b_1 - a_1 b_p \quad \cdots \quad b_{p-1} - a_{p-1} b_p\ ] \text{ where } b_i = 0 \text{ and } i \in [0, \dots, p-q]$$

where the coefficients a_i describe the characteristic polynomial of the model. This matrix shape is called a Controllable Canonical Form because the dynamics of the feedback are directly related to these coefficients. Specifically, given a feedback controller $\boldsymbol{K}$ for any particular model, the closed loop dynamics are

$$\boldsymbol{x}_{k+1} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})\boldsymbol{x}_k, \tag{2.39}$$

which translated to Reachable Canonical Form will have the shape

$$\boldsymbol{A}_c - \boldsymbol{B}_c\boldsymbol{K}_c = \begin{bmatrix} k_{n-1} - a_{n-1} & k_{n-2} - a_{n-2} & \cdots & k_1 - a_1 & k_0 - a_0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}, \tag{2.40}$$

which means we can directly modify each coefficient by selecting a controller parameter. In the MIMO case, the Controllable Canonical Form has the following shape:

$$\boldsymbol{A}_c = \begin{bmatrix} -\boldsymbol{a}_{p-1}\boldsymbol{M}_{p-1} & -\boldsymbol{a}_{p-2}\boldsymbol{M}_{p-2} & \cdots & -\boldsymbol{a}_2\boldsymbol{M}_2 & -\boldsymbol{a}_0\boldsymbol{M}_0 \\ \boldsymbol{I}_{n-1} & 0 & 0 & \cdots & 0 \\ 0 & \boldsymbol{I}_{n-2} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{I}_0 & 0 \end{bmatrix}, \boldsymbol{B}_c = \begin{bmatrix} \boldsymbol{M}_0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix},$$

$$\boldsymbol{C}_c = [\ \boldsymbol{\Gamma}_0 \quad \boldsymbol{\Gamma}_1 \quad \cdots \quad \boldsymbol{\Gamma}_{p-1}\ ], \tag{2.41}$$

with $\boldsymbol{I}_i, \boldsymbol{M}_i \in \mathbb{R}^{q_i \times q_i}, \boldsymbol{a}_i \in \mathbb{R}^{q \times q_i}, \boldsymbol{\Gamma}_i \in \mathbb{R}^o$ and for $i \in [0, \ldots, p-q], \boldsymbol{\Gamma}_i = 0$, where q is the number of inputs, q_i the number of inputs with a transfer function of order of at least i, $\boldsymbol{I}_i$ an identity matrix, $\boldsymbol{M}_i$ an upper triangular matrix with 1s in its diagonal, $\boldsymbol{a_i}$ a matrix of coefficients of the i^{th} order on its diagonal, and o the number of outputs. Ideally, $\forall i > 0, \boldsymbol{M}_i = \boldsymbol{I}_i$

Any LTI model can be transformed into a controllable canonical form by a matrix $\boldsymbol{T}_c$ such that $\boldsymbol{x} = \boldsymbol{T}_c\hat{\boldsymbol{x}}$ and $\boldsymbol{K} = \boldsymbol{K}_c\boldsymbol{T}_c$. Then the dynamics of the controllable form are

$$\hat{\boldsymbol{x}}_{k+1} = \boldsymbol{A}_c\hat{\boldsymbol{x}}_k + \boldsymbol{B}_c u_k, \text{ where } \boldsymbol{A}_c = \boldsymbol{T}_c\boldsymbol{A}\boldsymbol{T}_c^{-1} \text{ and } \boldsymbol{B}_c = \boldsymbol{T}_c\boldsymbol{B},$$

$$y_k = \boldsymbol{C}_c\hat{\boldsymbol{x}}_k + b_0 u_k, \text{ where } \boldsymbol{C}_c = \boldsymbol{C}\boldsymbol{T}_c^{-1},$$

where $\boldsymbol{T}_c$ can be calculated as follows:

Let

$$\boldsymbol{R} = [\ \boldsymbol{B} \quad \boldsymbol{A}\boldsymbol{B} \quad \boldsymbol{A}^2\boldsymbol{B} \quad \ldots \quad \boldsymbol{A}^{p-1}\boldsymbol{B}\ ] \tag{2.42}$$

be the reachability matrix of the model, and

$$\boldsymbol{R}_c = \begin{bmatrix} 1 & a_{p-1} & a_{p-2} & \dots & a_1 \\ 0 & 1 & a_{p-1} & \dots & a_2 \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \tag{2.43}$$

the reachability matrix of the canonical form. Then

$$\boldsymbol{T}_c = \boldsymbol{R}_c \boldsymbol{R}^{-1}. \tag{2.44}$$

2.6.3 Stability of a controlled system

There are a number of algorithms that can be used for stability analysis [26, 27].

A discrete-time LTI model as (2.39) is asymptotically stable if all the roots of its characteristic polynomial (i.e., the eigenvalues of the closed-loop matrix $A_d - B_d K$) are inside the unit circle of the complex plane, i.e., their absolute values are strictly less than one [19] (this simple sufficient condition can be generalised, however this is not necessary in our work). In this work, we express this stability specification $\phi_{stability}$ in terms of a check known as *Jury's criterion* [74]: this is an easy algebraic formula to select the entries of matrix K so that the closed-loop dynamics are shaped as desired.

Since the canonical forms used in this work use the coefficients of the characteristic polynomial $S(z)$ as opposed to its roots (which are comparatively hard to calculate for a SAT solver due to the number of multiplications involved), we choose Jury's criterion [19] to check the stability in the Z-domain in view of its efficiency.

We consider the following form for $S(z)$:

$$S(z) = z^p + a_1 z^{p-1} + \cdots + a_{p-1} z + a_p.$$

and build the matrix $\boldsymbol{M} \in \mathbb{R}^{p \times p}$ from its coefficients such that:

$$\begin{aligned}
\boldsymbol{M}_{1,j} &= a_j, \\
\boldsymbol{M}_{i+1,j} &= \begin{cases} 0, & j > p' \\ \boldsymbol{M}_{i,j} - \boldsymbol{M}_{i,p'-j+1} \frac{\boldsymbol{M}_{i,1}}{\boldsymbol{M}_{i,p'}} & j \leq p' \end{cases}, \\
p' &= p - i + 1.
\end{aligned}$$

$S(z)$ is the characteristic polynomial of a stable system if and only if the following conditions hold [19]:

$$\phi_{stability}(S) \equiv (S(1) > 0) \wedge ((-1)^p S(-1) > 0) \wedge (|a_p| < 1) \wedge \bigwedge_{i=1}^{p} (\boldsymbol{M}_{i,1} > 0).$$

2.6.4 Observable canonical form

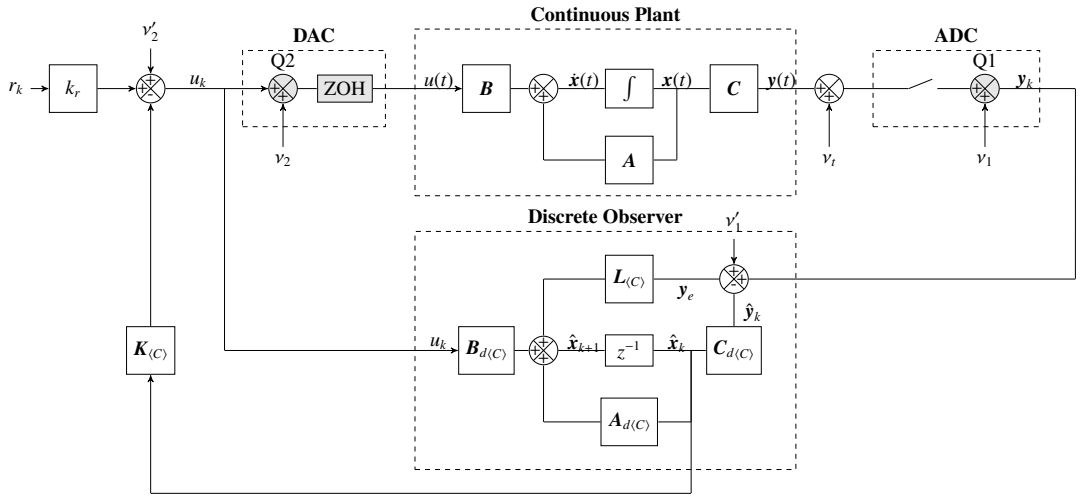

Figure 2.5: Closed-loop digital control interconnection with an observer, where the controller and the observer (bottom part) are implemented in a discrete domain, whereas the plant (top part) is continuous.

In most real implementations, the internal state of a plant cannot be directly measured but rather inferred from its effect on the output by using an observer. Essentially, a state observer is a model that provides an estimate of the internal state of a given real

system, from measurements of the input and output of the real system. This means that contrary to a stateless reactive controller, an observer controller has memory of the states (i.e. it is stateful).

Take for example a system where we are given the position of a robot arm in a single axis. Our objective is to move the arm to pick an object from a known location along that axis. The input of the robotic arm is an acceleration. The position of the arm will vary depending on the speed, which in turn varies with the acceleration. However, if we do not know the speed, we will not be able to reach our target smoothly. An observer for this design could simply calculate the speed by keeping a memory (state) of the previous position and calculate the difference with the current position in the given time interval.

A closed-loop digital control system with observer is shown in Figure 2.5. Note that the variables of a state observer are denoted by a "^" (e.g. $\hat{\boldsymbol{x}}(k)$) to distinguish them from the variables of the actual physical system. Moreover, $\boldsymbol{y}_e = \hat{\boldsymbol{y}}_k - \boldsymbol{y}_k$ is the output estimation error defined as the difference between the real output $\boldsymbol{y}_k$ and estimated one $\hat{\boldsymbol{y}}_k$, whereas $\boldsymbol{L}$ is the error sensitivity matrix, which amplifies the output estimation error. The figure also contains references to ν_1, ν_2, ν'_1, ν'_2 and ν_t, which denote noise and will be explained later in the dissertation. Lastly, k_r is a reference gain required by the model.

Presuming a sampled model of the plant as described in (2.11), the dynamics of the observer are:

$$\hat{\boldsymbol{x}}_{k+1} = (\boldsymbol{A}_{d\langle C\rangle} - \boldsymbol{L}\boldsymbol{C}_{d\langle C\rangle})\hat{\boldsymbol{x}}_k + \boldsymbol{B}_{d\langle C\rangle}\boldsymbol{u} + \boldsymbol{L}\boldsymbol{y}, \tag{2.45}$$

where the subindex $\langle C\rangle$ represents the domain of the controller $\mathbb{R}_{\langle C\rangle}$ such that $\boldsymbol{A}_{\langle C\rangle} = \mathcal{F}_{\langle C\rangle}(\boldsymbol{A}_d)$, $\boldsymbol{B}_{\langle C\rangle} = \mathcal{F}_{\langle C\rangle}(\boldsymbol{B}_d)$ and $\boldsymbol{C}_{\langle C\rangle} = \mathcal{F}_{\langle C\rangle}(\boldsymbol{C}_d)$. The differentiation between $*_{\langle C\rangle}$ and $*$ will be exploited later in this work. For the time being we will assume they are the same. Since the feedback of this system depends on $\boldsymbol{C}$ rather than $\boldsymbol{B}$, the Reachable Canonical form is no longer suitable. Hence we require an Observable Canonical Form. For SISO

systems, this is:

$$\boldsymbol{A}_o = \begin{bmatrix} -a_{p-1} & 1 & 0 & 0 & \cdots & 0 \\ -a_{p-2} & 0 & 1 & 0 & \cdots & 0 \\ -a_{p-3} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ -a_0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \boldsymbol{B}_o = \begin{bmatrix} \Gamma_{p-1} \\ \Gamma_{p-2} \\ \vdots \\ \Gamma_0 \end{bmatrix}, \tag{2.46}$$

$$\boldsymbol{C}_o = [\quad 1 \quad 0 \quad 0 \quad 0 \quad \cdots \quad 0\ ].$$

We notice here that $\boldsymbol{A}_o = \boldsymbol{A}_c^\intercal, \boldsymbol{C}_o = \boldsymbol{B}_c^\intercal$ and $\boldsymbol{B}_o = \boldsymbol{C}_c^\intercal$, which also applies to the MIMO case.

Let

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \boldsymbol{CA}^2 \\ \vdots \\ \boldsymbol{CA}^{p-1} \end{bmatrix} \tag{2.47}$$

be the observability matrix of the model, and

$$\boldsymbol{W}_o = \boldsymbol{R}_c^{-1} \tag{2.48}$$

the observability matrix of the canonical form (see eg [20]). Then the matrix $\boldsymbol{T}_o = \boldsymbol{W}^{-1}\boldsymbol{W}_o$ will transform an observable system into its canonical form. For simplicity of the description, we aggregate the noises as follows: $\eta_1 = \nu_1 + \nu_1' + \nu_t$ and $\eta_2 = \nu_2 + \nu_2'$, where ν_1 and ν_2 are the quantisation noises introduced by the ADC and the DAC, ν_t is the noise introduced by the delay, and ν_1' and ν_2' are quantisation noises caused by round-offs in the FWL operations (see Section 5.2). Assuming that the observer dynamics are an exact copy of the real ones, $(*_{\langle C \rangle} = *)$, the closed loop dynamics of this system are given by:

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}_e \end{bmatrix}_{k+1} = \begin{bmatrix} \boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K} & \boldsymbol{B}_d\boldsymbol{K} \\ \boldsymbol{0} & \boldsymbol{A}_d - \boldsymbol{L}\boldsymbol{C}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}_e \end{bmatrix}_k + \begin{bmatrix} \boldsymbol{B}_d k_r & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{L} \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_k + \boldsymbol{\eta}_2 \\ \boldsymbol{\eta}_1 \end{bmatrix}, \tag{2.49}$$

where $\boldsymbol{x}_e = \boldsymbol{x} - \hat{\boldsymbol{x}}$ is the estimation error of the state space by the observer. Note that this

matrix is not in canonical form. A transformation using both reachability and observability matrices is required. This is achieved as follows: By translating the coordinate system to

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{x}'_e \end{bmatrix} = \begin{bmatrix} \boldsymbol{T}_c & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T}_o \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}_e \end{bmatrix}, \tag{2.50}$$

where $\boldsymbol{T}_c = \boldsymbol{R}_c\boldsymbol{R}^{-1}$ and $\boldsymbol{T}_o = \boldsymbol{W}^{-1}\boldsymbol{W}_o$, we can transform Equation (2.49) into the equivalent canonical form

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{x}'_e \end{bmatrix}_{k+1} = \begin{bmatrix} \boldsymbol{A}_c & \boldsymbol{B}_o \\ \boldsymbol{0} & \boldsymbol{A}_o \end{bmatrix} \begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{x}'_e \end{bmatrix}_k + \begin{bmatrix} \boldsymbol{T}_c\boldsymbol{B}_d & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{T}_o\boldsymbol{L} \end{bmatrix} \begin{bmatrix} \boldsymbol{k}_r\boldsymbol{r}_k + \boldsymbol{\eta}_2 \\ \boldsymbol{\eta}_1 \end{bmatrix},$$

$$\boldsymbol{y}_k = \boldsymbol{C}_t\boldsymbol{x}_k, \tag{2.51}$$

where

$$\boldsymbol{A}_c = \boldsymbol{T}_c\left(\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K}\right)\boldsymbol{T}_c^{-1} \qquad \boldsymbol{A}_o = \boldsymbol{T}_o\left(\boldsymbol{A}_d - \boldsymbol{L}\boldsymbol{C}_d\right)\boldsymbol{T}_o^{-1}$$

$$\boldsymbol{B}_o = \boldsymbol{T}_c\boldsymbol{B}_d\boldsymbol{K}\boldsymbol{T}_o^{-1} \qquad \boldsymbol{C}_t = \boldsymbol{C}_d\boldsymbol{T}_c$$

Replacing the matrices in (2.51) with appropriate symbols:

$$\boldsymbol{A}' = \begin{bmatrix} \boldsymbol{A}_c & \boldsymbol{B}_o \\ \boldsymbol{0} & \boldsymbol{A}_o \end{bmatrix}, \qquad \boldsymbol{B}' = \begin{bmatrix} \boldsymbol{T}_c\boldsymbol{B}_d & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{T}_o\boldsymbol{L} \end{bmatrix}, \tag{2.52}$$

we get

$$\tilde{\boldsymbol{x}}_{k+1} = \boldsymbol{A}'\tilde{\boldsymbol{x}}_k + \boldsymbol{B}'\tilde{\boldsymbol{u}}_k \text{ where } \tilde{\boldsymbol{x}}_k = \begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{x}'_e \end{bmatrix}_k \text{ and } \tilde{\boldsymbol{u}}_k = \begin{bmatrix} \boldsymbol{r}_k + \boldsymbol{\eta}_2 \\ \boldsymbol{\eta}_1 \end{bmatrix}.$$

This has a similar structure to (2.9) which will be helpful in our analysis.

2.6.5 Stability of an observed system

Careful examination of the matrices of the observed system(2.52) reveals that because $\boldsymbol{A}'$ is upper block diagonal, the spectrum of $\boldsymbol{A}'$ is equal to the union of the spectrum of $\boldsymbol{A}_c$ and $\boldsymbol{A}_o$, which can be calculated separately. In other words, the characteristic polynomial of the closed loop is equal to the multiplication of the characteristic polynomials of the controller closed loop and the observer error when seen independently.

$$P(z) = P_K(z)P_L(z) = \left(z^p + \sum_{i=0}^{p-1}(a_i - k_i)z^i\right)\left(z^p + \sum_{i=0}^{p-1}(a_i - l_i)z^i\right).$$

We get the stability specification by applying Jury's Criterion to these polynomials.

2.7 Counterexample guided verification and synthesis

Since many of the abstractions described here are over- (or under-) approximations of the real dynamics, verification of these models may result in false positives (or negatives). i.e. a failed verification result does not necessarily imply that the original system does not satisfy the properties. In such situations we need to refine our abstractions.

2.7.1 CEGAR

Counter-example Guided Abstraction Refinement (CEGAR) [52] is the process by which a concrete trace of a system whose verification differs from an abstract verification result, thus revealing it as a false positive (or negative), is used as a means to refine the abstraction in order to obtain a verification result that extends to the original system, i.e. which is not a false positive (or negative). The true trace is called a counterexample.

Figure 2.6 shows a typical CEGAR loop. Initially, Abstract creates an abstraction of the system which is fed to Verify. If the verification is successful, we have a solution which we mark as a pass. Otherwise we validate the reported error to make sure it is not a spurious result. If the error exists in the concrete model, then we have a solution

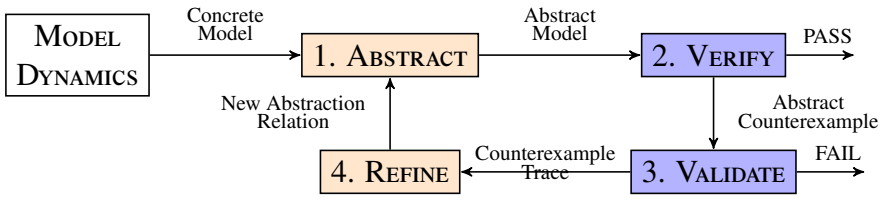

Figure 2.6: Block diagram describing a CEGAR loop. The orange(light) boxes correspond to abstraction phases and the blue(dark) to verification phases.

which we mark as a fail. If the validation is successful, then we refine the abstraction in order to get a valid result. This loop may go on indefinitely, which is why practical implementations also have a limit in the number of tries after which they will report the false positive if a proper refinement has not been found. We will also note that the Validate phase does not necessarily require verification against a concrete trace, but a more refined abstract trace may be used.

2.7.2 CEGIS

Counterexample Guided Inductive Synthesis (CEGIS) [154] is a formal method which synthesises a model given a specification using the aid of a verifier. In most cases the model is in the shape of a program (e.g. using syntax-guided synthesis [7] by defining a grammar for the program), whereas in the case analysed here it corresponds to a parametric set of values for a given shape of feedback dynamics, as described in Section 2.6. In the case of parametric synthesis, the purpose of the CEGIS model is to use an unsound synthesiser that searches the solution space (valuations of the parameters) to improve performance and validate its results with formal verification. The nature of the unsoundness comes from the fact that synthesis is performed to meet the specification given a subset of the valid inputs and may therefore not meet said specification for other valuations of the inputs. As with CEGAR, if the candidate solution generated by the synthesiser is not correctly verified against the original specification for a valuation of the inputs, a counterexample is generated which allows the synthesiser to include the valuations used by the counterexample in its next synthesis phase. As with CEGAR, the loop may have to exit prematurely to avoid infinite or extremely long loops.

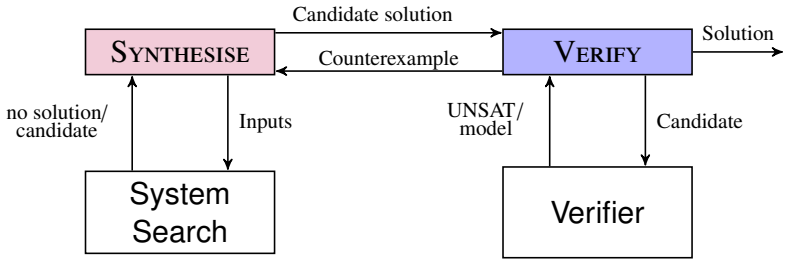

Figure 2.7: Counterexample-Guided Inductive Synthesis. The Synthesis phase is shown in red(light) and the verification phase in blue(dark)

Figure 2.7 shows a CEGIS loop. The SYNTHESISE phase explores a parameter valuation (which we refer to as a candidate solution) given a specification, and a set of counterexamples (initially empty). If a solution is found, it is only known to be valid for the selected values of inputs given by the counterexamples. The candidate solution is then passed to VERIFY, which checks the solution against all possible valuations of the input set. If a least one of these is invalid, it returns a new counterexample for the Synthesiser to use in its input valuation. If the Verifier is successful, then we have a solution. As stated before, this model may be extended to synthesise non-parametric solutions, but this case falls outside the scope of this work.

Chapter 3

Novelty and Related Work

3.1 Hybrid models

Hybrid models encompass all models that describe continuous and discrete behaviour. The differences between these two dynamics make it necessary for frameworks analysing them to acknowledge the nature of each independently and use methods to marry the differences in real-world cases. In the case of continuous behaviour, trajectories mark the pattern of development of a system, whereas in the discrete case, changes manifest as jumps from one state to another.

Although in this work we will not discuss the details of abstract acceleration within a hybrid model, it is worth mentioning that the technique lends itself to be embedded in models such as HIOA [126] (described in the next section). Some insight will be provided in the conclusions and future work, and we reference the literature here to contextualise it.

Early works on Hybrid Models focused on dividing the continuous space into finite states to approximate its behaviour using Finite Automata [142], which could manifest a state explosion even in the simplest systems with only a few variables. Several models arose with different purposes, amongst which Differential Automata [159] for modelling and simulation, and Sequential Control Automata [134] for digital control are worth of mention. Further work sought to unify these models into a single framework [37],

until eventually a modelling framework arose that accurately described such systems with simplicity and flexibility.

3.1.1 Hybrid automata

The limitations of the discretisation of the continuous space led to the development of Hybrid Automata (HA), initially proposed by [8] in the 1990s. The hybrid automaton develops over a hybrid space represented by a continuous state space ($\mathbb{R}^n$) and a set of discrete states (Q). Development along the continuous space is defined by a vector field, typically represented by differential equations, whereas development over the discrete space is represented by state transitions or jumps. In some cases, these jumps can happen along the continuous state space in which case they are called resets.

The model offers the advantage of accurately representing the dynamics of each state space whilst offering semantics for the interaction between the two. It is however a monolithic description of any system, thus lacking the expressiveness to accurately represent more structured representations (i.e. in this respect it is very rigid in its description). More specifically, it does not deal with external inputs or existentiality (which is fundamental to the analysis of chemical and software processes where elements of the system may not exist at all times), and it has no formalism for expressing modularity.

Several approaches have been proposed to overcome these limitations, amongst which we find compositional models such as Reactive Modules [10], Hybrid Behavioural Automata(HBA) [111] and Hybrid I/O Automata (HIOA) [126], and modelling languages such as SHIFT [66], CHARON [9] and PTOLEMY [73], which seek to exploit the difference in the inner structures of these automata without losing the formality of its expressions. In all of these approaches, the subject of modularity becomes the key to decomposing the behaviour into manageable fragments and a large part of the problem focuses then on the subject of composition, including issues of concurrency, serialisation, synchronisation and compatibility.

The HIOA model remains current largely because it addresses all these issues in a purely abstract manner without focusing on the internal mechanics of it components, but

it also suffers from a lack of concreteness for the same reason. Most approaches along this line still use the original hybrid automata semantics in order to describe the inner behaviour of an HIOA, which retains its limitations to express structured software models. One of the greatest difficulties found is the need to solve non-linear equations, more specifically combinations of transcendental and polynomial ones. For this reason, most of the work focuses on Rectangular Automata (RHA) [99], where all variables are pairwise independent or, more typically, Linear Hybrid Automata (LHA) [99] where ODEs are linear with respect to the derivatives of the variables of interest (though the overall behaviour may not be linear itself). A common approach for more complex automata is to approximate them to one of the former (RHA or LHA). If the approximation holds, then the system can be treated using the dynamics of these automata. There are several methods for solving linear equations and box invariance, thus the problem becomes one of efficiency. However, modern techniques have allowed the possibility for more complex representations which will be explored in this work.

The HA framework (and its successors) is still the most widely used reference for Hybrid system analysis, either as the base structure for modern techniques or models [25, 110, 127], or as the benchmark that new techniques are measured against [88, 139]. The equations are often varied (and on occasion the semantics slightly adjusted), but the framework has become the paradigm for hybrid system modelling.

The latest work on this area refers to Hybrid Automata Networks, which is the composition of modular HAs (such as HIOA), and is focusing on the automation and scalability of verification algorithms. Given recent developments in the scalability of Satisfiability solvers (SAT), a popular approach is the inclusion of Satisfiability Modulo Theories (SMT) on Hybrid Automata [50, 110] due to their ability to compute a very large number of formulas in reasonable times.

3.2 Reachability analysis

Traditionally there are two broad approaches to the reachability problem that we seek to analyse.

3.2.1 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. Bounded-time reachability of discrete-time LTI systems 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 explicitly considered, and may thus violate a safety requirement. These solutions may be extended to an unbounded time horizon if a fixpoint is found, proving that the progression from a given point in time is included in the reach set evaluated up until that point. Representatives are STRONG [65], HySon [34], CORA [5], HYLAA [22] and SpaceEx [83]. A comparison of our technique with some of these tools can be found in [6].

Set-based simulation methods generalise guaranteed integration of ODEs [33, 125], 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 [100] and PHAVer [82]), polyhedral flow-pipes [48], ellipsoids [32], and zonotopes [91]. We note that tools such as PHAVer handl the unbounded case by checking for a fix-point as explained above (*i.e.* that the reachable space is strictly contained in the analysis region). A breakthrough has been achieved by [92, 96], with the representation of convex sets using template polyhedra and support functions. This method is implemented in the tool SpaceEx [83], which can handle dynamical systems with hundreds of continuous 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 [69]) 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 [22], which is the first tool to solve all high-order problems described in [161] which include several with hundreds of state variables. Many of these approaches calculate explicit reachability at given iterations, therefore providing more precision and detail than our proposed approach. Only one tool that we are aware of in this field offers explicit assurances about the numeric soundness of the result: Flow* [46] uses interval arithmetics in its calculations to ensure soundness.

Other approaches focus on bounded model checking of hybrid automata, and use specialised constraint solvers (HySAT [81], iSAT [72]), or SMT encodings [51, 97].

novelty Although we are limited to linear continuous dynamics, our novelty over these approaches relates to soundness. First because we are capable of reasoning about unbounded time horizons whilst retaining scalability, and secondly because we find bounds in our numeric calculations that ensure numeric soundness.

3.2.2 Unbounded time reachability analysis

The second approach, epitomised by static analysis methods [98], explores unbounded-time horizons. It employs conservative over-approximations to achieve relative completeness (ensure 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 [110], Sting [56], and abstract interpreters such as Astrée [29] and InterProc [107]. Early work in this area has used implementations of abstract interpretation and widening [57], which represent still the underpinnings of many modern tools. The work in [98] 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 [31] following an inductive sequence of half-space interpolants. [59] employ optimisation-based (max-strategy iteration) with linear templates for hybrid systems with linear dynamics. Relational abstractions [149] use ad-hoc "loop summarisation" of flow relations, while abstract acceleration focuses on linear relations analysis [93, 94], which is common in program analysis.

novelty The work in [108] shows the advantages of abstract acceleration against static analysis tools. Our novelty in this respect comes from the fact that we have extended the capabilities of the technique by making it faster, more precise, and capable of dealing with a wider range of systems. Details about this novelty will be described in the next section.

3.2.3 Abstract acceleration

Abstract acceleration [93, 94, 108] 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 [153] but restricted to subclasses of linear loops, and later to general linear loops but without inputs [108].

The work presented in this thesis lifts these limitations by presenting abstract acceleration for *general* linear loops *with* inputs [42, 43], developing numeric techniques for scalability and extending the domain to continuous time models. It addresses several of the challenges presented in [152] which briefly describes the state of the art of abstract acceleration, its application to Hybrid Automata (which is briefly described both there and in this work), and the requirement for higher precision (which is achieved in our work through CEGAR).

The work in [122] puts forward improvements on [42], and its outcomes can be compared with the results in [43, 44]. Indeed, [122] 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 [43]. 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 [43], thus a combined approach may be more efficient. The handling of the guards presented in [122] has a number of advantages over our approach. The first is that their description is mathematical rather than procedural, hence more tractable. The second is that our procedure is not complete, resulting on occasion in an inability to find a reasonably precise bound. However, in the typical case our procedure works well and in the experiments we have run, it has always been able to find the optimal solution.

novelty Our first claim of novelty relates to precision. By using support functions instead of logahedral abstractions, we allow ourselves more freedom in the choice of template directions, which in turn enables tighter over-approximations. Furthermore, we use CEGAR to choose the best directions to lift a given violation which is suspected to be caused by the over-approximation.

Our second claim is related to speed performance. By using numeric techniques we are able to increase the speed of the calculations by several orders of magnitude at a comparatively minimal cost in precision. The cost of restoring soundness to the numeric calculations is also considerably smaller than the speed-up achieved by their introduction. It is worth noting that the contributions made in this respect relate to other fields in verification, where we extend this claim of novelty.

Finally, we have extended the capabilities of abstract acceleration to problems that are by their nature unsolvable with the pre-existing technique. In particular for the case of systems with non-deterministic time-variable inputs and systems with continuous dynamics.

3.3 Simplex implementations

There are historically two decision procedures that are commonly used for linear arithmetics: Fourier-Mozkin elimination and the simplex method. SMT solvers prefer the latter because of its better performance when using rational arithmetics, whilst linear abstraction domains favour it for its performance regarding linear optimisation problems. The original simplex has also been updated with a revised version based on its duality which allows for correction of accumulated errors in the case of unsound or imprecise methods. Although some tools favour unsound implementations, there are cases where the need for soundness eliminates the possibility of using floating point as a means to perform fast linear program analyses. Two solutions have arisen to tackle this problem. The first one [131] uses a floating point simplex to traverse the vertices of a polyhedron and then follows the selected pivots by forcing a rational simplex along the same path. If the original method is unsound, the rational simplex continues its path (thus losing the speed advantage) until it finds the correct solution. The second method [45] uses interval arithmetics on Fourier-Motzkin elimination to over-approximate the solution for abstract polyhedral domains. Since abstract domains are typically over-approximations of the concrete domains they abstract, exact solutions are not necessary, and these over-approximations are acceptable. Our novelty in this respect is that we apply interval arithmetics to the simplex procedure directly, which results in an efficient implementation for linear optimisation. These two methods [45, 131] show an improvement of the floating point performance of 3:1 -10:1 in most case studies. However, the analysis done in [63] shows that for a large number of real world programs evaluated by openSMT an unsound floating point simplex is marginally faster ($\tilde{1}3\%$) than an exact rational simplex . This is due to some optimisations in the handling of rationals inside the SMT solver and the fact that it is only checking for feasibility as opposed to optimality.

novelty Existing approaches will yield in most circumstances faster results than our method, since the rational number explosion remains small due to optimisation. However, they are not capable of reasoning about problems where errors have been introduced in

the preprocessing phase (as is the case when eigendecomposition is performed), which is our main contribution in this regard.

3.4 Controller synthesis

3.4.1 Static output feedback

The problem of stability is often studied as Static Output Feedback stabilisation (SOF). Most techniques for control verification in this field look at Lyapunov stability [3, 113, 118, 143, 148]. The problem is translated into a Linear Matrix Inequalities (LMI) problem, which may be solved iteratively (ILMI [40]), or using Bilinear Matrix Inequality solvers (BMI [77]). Other techniques use H_∞ [3, 70] and H_2 [16] norms to achieve stability, whilst [47] uses polynomial function optimisations to the same effect. The non-smooth approach described in [15] uses H-norms and other parameters independently to tune the controller. The Ray-Shooting approach from [138] uses direct pole placement in two stages (one for feasibility, one for optimisation). Several researchers use Moore's algorithm [114] to this effect, focusing on various optimisations to improve performance [117, 151], particularly in the case of MIMO systems where there are not unique solutions for the synthesis matrices [128]. In some cases, restrictions are placed on the control structure, such as in the Internal Model Principle (IMP) [79], which replicates the dynamics of persistently exciting inputs (*i.e.* any sequence of inputs that leads to a variation in the output) in the structure of the controller by placing linear constraints or even zero coefficients in the final model. Several papers look at trading multiple specifications, for example in the case of simultaneous stabilisation [38] or mixed H_2/H_1 controllers [17]. Robust control is equivalent to extending simultaneous stabilisation to an infinite number of plant models (typically corresponding to tolerances in the dynamics and/or noise in the signal transfer). The state of the art in this area looks at polytopic descriptions [67]. As stated in the introduction, due to the frequency domain approach used in most of these analyses, transitive response and exogenous inputs are not considered in these papers.

Focusing on the controller stability problem, specifically on the effects of digital code on system robustness, more recent formal tools rely on an invariant computation on the discrete system dynamics using Semi-Definite Programming (SDP). These solvers use floating point arithmetic and the error is upper bound to establish soundness. [164] looks at Bounded Input/Bounded Output stability, whilst [146] uses Lyapunov-based quadratic invariants. Some of the caveats of these tools are their focus on closed systems with set points, and lack of examination of transient states.

3.4.2 Parameter synthesis

State space based parameter synthesis has historically used reachability tools for hybrid linear [84, 101] and non-linear systems [68], but they remain restricted in their capabilities. Since they are typically based on simulation runs, most of them are restricted to bounded-time analysis. Other tools [49, 84] use counterexample guided exploration to find the set on non-viable solutions, and by complement the feasible parameter set. An alternative approach is to synthesise viable parameters from an initial finite valuation of the parameter set, and by finding new parameters that result in similar system runs as the ones initially supplied [12, 13]. Whilst these approaches have covered the problem for non-linear and hybrid systems, only few claim to find maximal sets [49].

[68] uses sensitivity analysis to calculate reach sets of otherwise intractable non-linear hybrid automata from individual time-bounded set-based simulation runs. The algorithm uses an error control mechanism to obtain a desired precision, but is otherwise unable to ensure an under-approximation of the parameter set. It also uses a representation that is akin to zonotopes, and therefore not as tight as our polyhedral domains, further reducing its ability to find optimal sets for our use case. [84] finds suitable but not optimal under-approximations of a parametric input set for LHA using counterexample-guided abstraction refinement (CEGAR) on discrete abstractions of the original system. Its description is limited to rectangular and octagonal sets. [49] uses CEGAR on inductive invariants of the reach set and estimates the parameter space as the complement of the counterexamples found by an SMT solver. Unlike the previously described techniques,

their algorithm claims to be precise thus finding a maximal solution for the problem in a given abstraction. Using modern SAT/SMT solvers, tools for inductive synthesis [62] can extract parameters with complex restrictions. It is important to note that tools capable of synthesising parameters for non-linear systems, are by definition able to synthesise the dynamics of an LTI model (the elements of $\boldsymbol{A}$ in a closed loop system). However, these results are somewhat curbed in the case of variable inputs due to state-space explosion.

Finally there are tools aimed at synthesising variable program inputs [4] with the purpose of finding a sequence that meets a desired goal. In this case the restriction on the results is that they are existential but not generalised, thus they do not serve the safety property.

3.4.3 Digital control synthesis

Program synthesis is the problem of computing correct-by-design programs from high-level specifications. Algorithms for this problem have made substantial progress in recent years, for instance [106] to inductively synthesise invariants for the generation of desired programs.

Program synthesisers are an ideal fit for the synthesis of digital controllers, since the semantics of programs capture the effects of finite-precision arithmetics precisely. In [143], the authors use CEGIS for the synthesis of switching controllers for stabilizing continuous-time plants with polynomial dynamics. The work extends to affine systems, but is limited by the capacity of the state-of-the-art SMT solvers for solving linear arithmetics (e.g. most SMT solvers reason about feasibility rather than optimality, thus providing suboptiomal results). Since this approach uses switching models instead of linear dynamics for the digital controller, it avoids problems related to finite-precision arithmetics, but potentially suffers from state space explosion. Moreover, in [144] the same authors use a CEGIS-based approach for synthesizing continuous-time switching controllers that guarantee *reach-while-stay* properties of closed-loop systems, i.e., properties that specify a set of goal states and safe states (constrained reachability). This solution is based on synthesizing control Lyapunov functions for switched systems that

yield switching controllers with a guaranteed minimum dwell time in each mode. However, both approaches are unsuitable for the kind of control we seek to synthesise (*i.e.* minimal states PID-type controllers).

The work in [1] synthesises stabilizing controllers for continuous plants by exploiting bit-accurate verification of software-implemented digital controllers [27, 105] but it is restricted to closed-loop systems given as transfer function models. By contrast, [2] considers a state space representation of the physical system, which has known advantages over the transfer function representation [20]: (i) it generalises to multivariate systems (i.e., with multiple inputs and outputs); (ii) gives the state-to-input relationships, offering an internal model of the system; and (iii) allows to synthesise control systems with guarantees on the internal dynamics, e.g., to synthesise controllers that make the closed-loop system *safe*. Our work focuses on the *safety* of internal states, which is usually overlooked in the literature. Moreover, this work integrates an abstraction refinement step inside the main CEGIS loop.

A large body of work focuses on the synthesis of 'Reactive Controllers' (also called 'Symbolic Controllers' e.g. [133]) which is a term coined for the synthesis of digital controllers using LTL specifications over discrete abstractions of plants. As an example of these, the tool Pessoa [129] synthesises correct-by-design embedded control software in a Matlab toolbox. It is based on the abstraction of a physical system to an equivalent finite-state machine and on the verification of reachability properties on it. Based on this safety specification, Pessoa can synthesise embedded controller software for a range of properties. The embedded controller software can be more complicated than the state-feedback control we synthesise, and the properties available cover more detail. Along this research line, [14, 124] study the synthesis of digital controllers for continuous dynamics, and [165] extends the approach to the recent setup of Network Control Systems.

novelty Relying on state space discretisation is likely to incur scalability limitations due to state space explosion: the work in [145] summarises the state of the art for these systems: although limitations to bounded time have been lifted through the use of abstractions of the LTL model and further refinement to enable adequate control of the

concrete system, the refinement loops have proven to be cumbersome, suffering from state space explosion in their implementation. 'Given the fact that an abstraction may very well comprise millions of states and billions of transitions, an implementation of the refined controller is often too expensive to be practical.' [145]. In some cases, we must also indicate that soundness is already an issue since the LTL model is already an abstraction of the continuous plant, and controllers synthesised for this model do not necessarily extend safely to the continuous case (except when explicitly formally verified for this case). Our approach by comparison requires a minimal description of the plant, and minimal implementation which is sound with respect to the continuous-discrete hybrid nature of the closed loop system. This is largely because we analyse the system over the dynamics rather than the states, and because the CEGIS loop includes a sound verification phase over both time domains. Furthermore, our optimisation refinement approach to CEGIS, enables us to synthesise controllers faster than using a naive approach, in many cases enabling the feasibility of solutions that the naive approach is not capable of producing in reasonable times.

Chapter 4

General Abstract Acceleration with Inputs for Safety Checking

4.1 General abstract acceleration with inputs

We verify safety by exploring the reachable set of the system over an unbounded time horizon. Our algorithm therefore targets the reachability analysis of the initial set using Abstract Acceleration. We will describe next the techniques used to achieve this goal first for discrete time systems and later for continuous time ones. The objective of the algorithm is to find a tight over-approximation of the transitive closure (referred here as the reach tube) of the system progression given an initial set of states.

4.1.1 Overview of the algorithm

The basic steps required to compute a reach tube using abstract acceleration are shown in Figure 4.1.

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 estimations identified by a hat ($\hat{\boldsymbol{S}}, \hat{\boldsymbol{J}}$).

2. The second step involves upper-bounding the rounding errors in order to obtain sound results: 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, identified by blackboard bold symbols.

3. The inverse of the generalised eigenvectors ($\mathbb{S}^{-1}$) is calculated soundly (see [44]).

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:

$$X'_k = \mathbb{J}\left(X'_{k-1} \cap G'\right) + U' \tag{4.1}$$

   where $\quad X'_k = \mathbb{S}^{-1} X_k, \quad U' = \mathbb{S}^{-1} \boldsymbol{B} U, \quad$ and $\quad G' = \{\boldsymbol{x} \mid \boldsymbol{G}\mathbb{S}\boldsymbol{x} \leq \boldsymbol{h}\}$.

5. We calculate the number of iterations n based on the guard set, as explained in Section 4.2. 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 or no inputs this step will be ignored) using the techniques described in Section 4.1.5.

7. We calculate the abstract dynamics using the techniques described in Section 4.1.2.

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 [44] to evaluate the convex set Hadamard product of the abstract dynamics (used as the tableau) and the initial set (whose vertices are used as the objective functions alongside a set of template directions for the result).

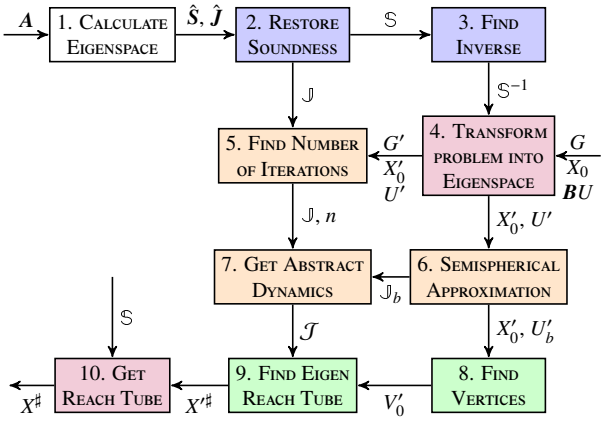

Figure 4.1: Block diagram describing the different steps used to calculate the abstract reach tube of a model via AA. The white box is the numerical eigensolver stage. Blue boxes are soundness restoration stages. Red boxes represent linear transformations of the problem. The orange boxes denote abstractions 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 original state space via multiplication by $\mathbb{S}$. It is worth noting that since our initial solution is an over-approximation of the solution in the convex spaces, this back transformation will also apply to the errors introduced by the over-approximation which can be seen as being multiplied by the condition number $k(\mathbb{S})$, hence resulting in a much looser over-approximation.

4.1.2 Using support functions for abstract acceleration

As an improvement over [108], the rows in $\boldsymbol{\Phi}$ and $\boldsymbol{f}$ (see (2.32)) can be obtained by better sampling of the support functions in these sets. The choice of directions for these support functions results in an improvement over the logahedral abstractions used in previous work [93, 94, 108] (see Figures 4.2 - 4.5). This 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 support function for the corresponding hyperplane. A third point taken from the curve is used to test the direction of the

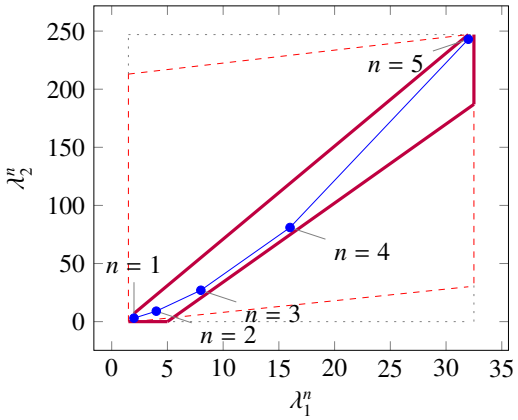

Figure 4.2: Polyhedral faces over $\mathbb{R}^2$ for pairs of eigenvalues $(\lambda_1^n, \lambda_2^n)$ where λ_1=2, λ_2=3, and $1 \leq n \leq 5$. Bold purple lines represent supports. The dotted grey and dashed red polytopes show logahedral approximations (box and octagon) used in [108]. Note the scales (the sloped dashed lines are parallel to the x=y line, and the dashed red polytope hides two small faces yielding an octagon).

corresponding template vector. An arbitrary number of additional hyperplanes are created by selecting pairs of adjacent points in the curve and creating the corresponding support functions, as shown in Figure 4.2.

2. Complex conjugate eigenvalue pairs

   In the case of complex conjugate pairs, the eigenvalue map corresponds to a logarithmic spiral (See Figure 4.3). In this case, we must first extract the number of iterations (denoted by $\bar{k}$) required for a full cycle. For convergent eigenvalues ($\|\lambda\| < 1$), only the first $\bar{k}$ iterations have an effect on the support functions, while in the divergent case only the last $\bar{k}$ iterations are considered (since symmetrically, it corresponds to the reverse spiral case). Support functions are found for adjacent pairs, checking for the location of the first point for convergent eigenvalues, and of the last point for divergent eigenvalues. If such point falls outside of the supporting half-space, we look for an interpolant point that closes the spiral 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 4.3). 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

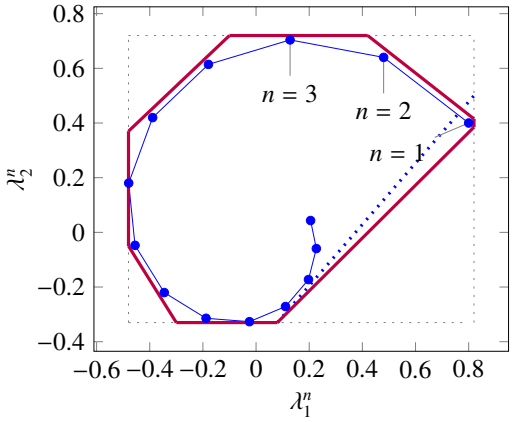

Figure 4.3: Polyhedral faces over $\mathbb{R}^2$ for complex conjugate eigenvalues $(\lambda_1^n, \lambda_2^n)$ where λ_1=0.8 + 0.4i, λ_2=0.8 − 0.4i, and 1≤n≤14. Bold purple lines represent supports.The blue dotted line shows the support function that excludes the point obtained with $n = 1$, which is replaced by the support function projecting from said origin.

$f = \max\left(\{1, |\lambda|^{\hat{n}} \cos(\theta)^{-1}\}\right)$, 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 $\cos\left(\frac{\theta}{m}\right)^{-1}$.

3. Equal eigenvalues

   When two eigenvalues are the same, the resulting support functions are those orthogonal to the $x = y$ plane, intersecting the square created by the maximum and minimum values (alternatively, a space reduction can be applied to remove the duplicate, and the original space restored after the reach tube has been evaluated).

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 convexity can be sharp, it is important to find the apex of the upper diagonals in order to minimise the over-approximation. See Figure 4.4. When both elements are divergent, the shape is similar to a positive valued pair since there is no apex. 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.

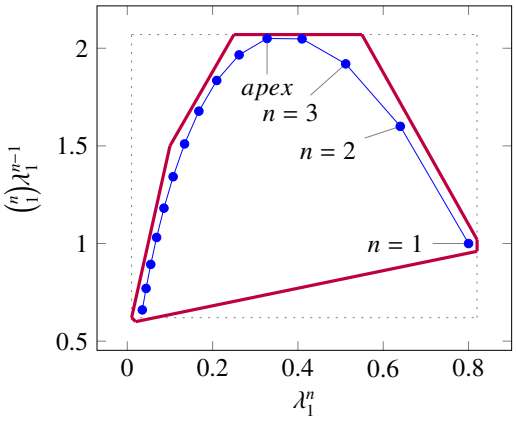

Figure 4.4: Polyhedral faces over $\mathbb{R}^2$ related to a Jordan block $(\lambda_1^n, \binom{n}{1}\lambda_1^{n-1})$ where λ_1=0.8 and $1 \leq n \leq 15$. Bold purple lines represent supports found in this work.

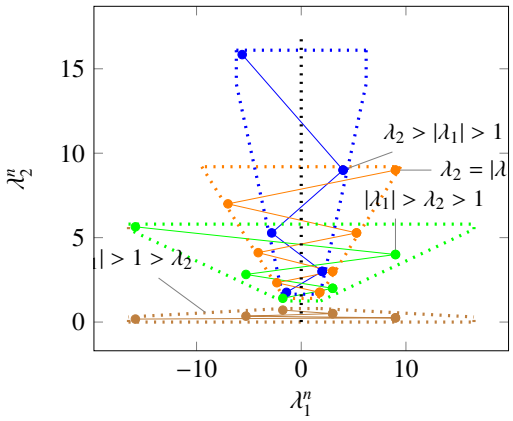

Figure 4.5: Polyhedral faces over $\mathbb{R}^2$, of different eigenvalue ratios (note that the curves obtained from the progression of the blue and orange dots are convex w.r.t. the λ_2^n-axis, whereas the green and brown are concave). Dotted lines represent convex supports for these layouts.

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 4.5). In the case of complex eigenvalues, we use the absolute value in order to determine the envelope. If both eigenvalues 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 [108] 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 $\delta_{max} = \|\mathbb{A} - \hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}^{-1}}\|$, where $\hat{\mathbb{J}}$ and $\hat{\mathbb{S}}$ are the eigenvalues and eigenvectors of $\boldsymbol{A}$ [44] calculated numerically (see Section 6.1.2 for details). 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 $|\lambda_s \pm \delta_{max}|^k$ (represented in Figure 4.2 with diameters around the blue dots).

The outward relaxation of the support functions ($\boldsymbol{f}$) follows a principle similar to that in [89], 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 the polyhedra outward, which has 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 overapproximations due to the condition number of the eigenvectors.

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 [44].

4.1.3 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 (2.32) we have

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

which leads to

$$\rho_{\mathcal{A}X}(\boldsymbol{v}) = \rho_{\boldsymbol{S}\varphi(\boldsymbol{m})\boldsymbol{S}^{-1}X}(\boldsymbol{v}) = \rho_{\varphi(\boldsymbol{m})\boldsymbol{S}^{-1}X}\left(\boldsymbol{S}^{\intercal}\boldsymbol{v}\right), \tag{4.2}$$

where

$$\rho_{\varphi X}(\boldsymbol{v}) = \sup\left\{\rho_{\varphi}(\boldsymbol{x} \circ \boldsymbol{v}) \mid \boldsymbol{x} \in X\right\}, \tag{4.3}$$

and

$$\rho_{\varphi}(\boldsymbol{v}) = \sup\{\boldsymbol{m} \cdot \varphi^{-1}(\boldsymbol{v}) \mid \boldsymbol{\Phi m} \leq \boldsymbol{f}\}. \tag{4.4}$$

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

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

and in order to simplify the nomenclature we write

$$\rho_{\mathcal{A}X}(\boldsymbol{v}) = \rho_X(\mathcal{A}^{\intercal}\boldsymbol{v}). \tag{4.5}$$

4.1.4 Acceleration of parametric inputs

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

$$\tau^{\sharp}(X_0, U) = \boldsymbol{A}X_0 \oplus \boldsymbol{B}U, \tag{4.6}$$

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 (2.3) (ignoring the presence of the guard set G for the time being), we obtain

$$X_n = \boldsymbol{A}^n X_0 \oplus \sum_{k=0}^{n-1} \boldsymbol{A}^k \boldsymbol{B}U. \tag{4.7}$$

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:

$$(\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1} = \boldsymbol{S}\boldsymbol{D}^n\boldsymbol{S}^{-1}, \tag{4.8}$$

given

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

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

$$d(\lambda_i, n, k) = \frac{-1^k}{k+1}\frac{1-\lambda_i^n}{(1-\lambda_i)^{k+1}} + \sum_{j=1}^{k} \frac{-1^{k-j}}{k-j}\binom{n}{j-1}\frac{\lambda_i^{n-j-1}}{(1-\lambda_i)^{k-j}},$$

where $gm(\cdot)$ is 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*

$$\hat{\tau}^{\sharp n}(X_0, U) =_{\text{def}} \mathcal{A}^n X_0 \oplus \mathcal{B}^n U, \tag{4.9}$$

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 (2.4) we have

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

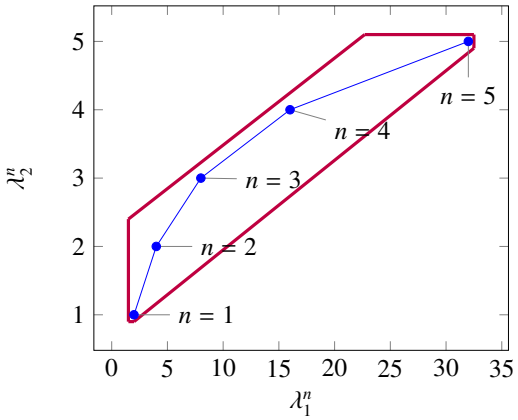

Figure 4.6: Polyhedral faces (λ_1^n, n) over $\mathbb{R}^2$, where λ_1=2, λ_2=3, and $1 \leq n \leq 5$. Bold purple lines represent supports found in this work.

Using equation (4.7), 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

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

$\mathcal{A}^n$ and $\mathcal{B}^n$ are calculated using the techniques described in Section 4.1.2 where special consideration is taken to evaluate the pairs with a unitary eigenvalue. Figure 4.6 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 4.1.2) therefore stand. □

4.1.5 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

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

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

$$\boldsymbol{u}'_{ci} = \frac{1}{2}(\rho_{U'_J}(\boldsymbol{v_i}) + \rho_{U'_J}(-\boldsymbol{v_i})), \text{ where } \boldsymbol{v}_{ij} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}. \tag{4.10}$$

We then over-approximate U'_d via U'_b, by the maximum radius in the directions of the complex eigenvalues (cf. illustration in Figure 4.7). 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 0s in $\boldsymbol{v}_b$ such that if $\boldsymbol{v}_b = [\ v_1 \quad 0 \quad v_3 \quad \dots \quad v_p\ ]^\intercal$ then $red(\boldsymbol{v}_b) = [\ v_1 \quad v_3 \quad \dots \quad v_p\ ]^\intercal$). 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

$$U'_d = \{\boldsymbol{u} \mid \boldsymbol{C}'_u \boldsymbol{u} \leq \boldsymbol{d}'_u\}, U'_d \subseteq U'_b, \text{ with}$$

$$U'_b = \{\boldsymbol{u} \mid F_b(\boldsymbol{C}'_u) f_b(\boldsymbol{u}) \leq f_b(\boldsymbol{d}'_u)\}, \text{ and}$$

$$\boldsymbol{B}U \subseteq U_b \oplus U_c, \text{ where } U_b = \boldsymbol{S}U'_b \text{ and } U_c = \{\boldsymbol{S}\boldsymbol{u}'_c\}. \tag{4.11}$$

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,

$$\boldsymbol{J}_b = \begin{bmatrix} \boldsymbol{J}_{b1} & & \\ & \ddots & \\ & & \boldsymbol{J}^n_{br} \end{bmatrix}, \tag{4.12}$$

$$\text{where } \forall s \in [1, \dots, r], \sigma(\boldsymbol{J}_{bs}) = |\sigma(\boldsymbol{J}_s)| \text{ and } gm(\boldsymbol{J}_{bs}) = gm(\boldsymbol{J}_s)^1,$$

which is equivalent to removing the conjugate pair with the negative imaginary part and using the norm of the eigenvalues of the reduced matrix.

Definition 4. *Given a matrix $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}$ and a vector $\boldsymbol{x}$, we define the following operations:*

$$F^*_b(\boldsymbol{A}, \boldsymbol{x}) = \boldsymbol{S} f_b^{-1}\left(F_b(\boldsymbol{J}) f_b(\boldsymbol{S}^{-1}\boldsymbol{x})\right). \tag{4.13}$$

Finally, we refer to the accelerated sets

$$U^n_b = \left\{F^*_b((\boldsymbol{I} - \boldsymbol{A}^n), F^*_b((\boldsymbol{I} - \boldsymbol{A})^{-1}, \boldsymbol{u})) \mid \boldsymbol{u} \in U_b\right\},$$

$$U^n_c = (\boldsymbol{I} - \boldsymbol{A}^n)(\boldsymbol{I} - \boldsymbol{A})^{-1} U_c, \quad U^n_{cb} = U^n_c \oplus U^n_b.$$

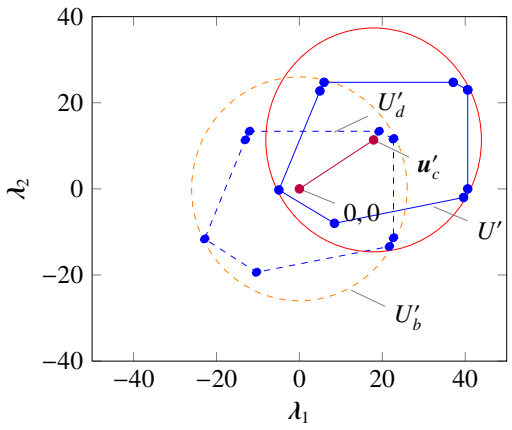

Figure 4.7: Relaxation of an input set within a complex subspace, in order to make it invariant to matrix rotations. Dashed lines and curves denote translated quantities onto the origin.

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

$$X_n \subseteq \boldsymbol{A}^n X_0 \oplus U^n_{cb}. \tag{4.14}$$

Shifting our attention from reach sets to reach tubes, we can now over-approximate the reach tube by abstract acceleration of the summands in (4.14), as follows.

Theorem 3. *The abstract acceleration*

$$\hat{\tau}^{\sharp n}(X_0, U) = \mathcal{A}^n X_0 \oplus \mathcal{B}^n U_c \oplus \mathcal{B}^n_b U_b \tag{4.15}$$

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

Proof. From Equation (4.14) we have $X_n \subseteq \boldsymbol{A}^n X_0 \oplus U^n_{cb} = \boldsymbol{A}^n X_0 \oplus U^n_c \oplus U^n_b$. Furthermore, from Equation (4.14) we also have $\hat{\tau}^{\sharp n}(X_0, U_c) \supseteq \bigcup_{k\in[1,\dots,n]} \boldsymbol{A}^k X_0 \oplus U^k_c$. Finally, from the definition of $\mathcal{B}^n_b$ we have $\mathcal{B}^n_b U_b \supseteq \bigcup_{k\in[1,\dots,n]} U^k_b$, hence $\hat{\tau}^{\sharp n}(X_0, U) \supseteq \hat{X}_n$. □

[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 $U'_b = \boldsymbol{S}^{-1} B U_b$, $U'_c = \boldsymbol{S}^{-1} B U_c$ and $U'_d = \boldsymbol{S}^{-1} B U_d$. Hence, this inclusion is also valid in the original state space.

4.1.6 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 of $\mathcal{B}_b^n$ this relationship is not linear (see Eq. 4.11). 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 $\rho_{X_{k+1}}(\boldsymbol{v}) = \rho_{\boldsymbol{A}X_k}(\boldsymbol{v}) + \rho_{\boldsymbol{B}U}(\boldsymbol{v})$, we define a set

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

so that

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

Accelerating X_{k+1}, we obtain

$$\rho_{X_n}(\boldsymbol{v}) = \rho_{\boldsymbol{A}^n X_0}(\boldsymbol{v}) + \rho_{(\boldsymbol{I}-\boldsymbol{A}^n)(\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}U}(\boldsymbol{v}) = \rho_{X'_0} \left((\boldsymbol{D}^n)^\intercal \boldsymbol{v}' \right),$$

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

$$\mathcal{D}^n = \begin{bmatrix} \mathcal{A}^n & 0 \\ 0 & \mathcal{B}^n \end{bmatrix}, \text{ so that } \rho_{X_n}(\boldsymbol{v}) = \rho_{X'_0} \left((\mathcal{D}^n)^\intercal \boldsymbol{v}' \right), \tag{4.16}$$

where $\mathcal{A}^n$ and $\mathcal{B}^n$ are the abstract matrices in Equations (2.32) and (4.9). We can then apply this abstraction to (4.11) and obtain:

$$\rho_{X_n}(\boldsymbol{v}) = \rho_{X'_0}(\mathcal{D}_b^{nT}\boldsymbol{v}'), \text{ where} \tag{4.17}$$
$$\mathcal{D}_b^n = \begin{bmatrix} \mathcal{A}^n & 0 \\ 0 & \mathcal{B}_b^n \end{bmatrix}, \quad \boldsymbol{v}' = \begin{bmatrix} \boldsymbol{v} \\ f_b(\boldsymbol{v}) \end{bmatrix},$$
$$\mathcal{B}_b^n = \boldsymbol{S}F_b^{-1}\left((\boldsymbol{I} - \mathcal{J}_b^n)(\boldsymbol{I} - \boldsymbol{J}_b)^{-1}F_b(\boldsymbol{S}^{-1})\right),$$

with $\boldsymbol{J}_b$ defined in (4.12). This model provides a tighter over-approximation than (4.15), 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* $\hat{\boldsymbol{x}} = \bigcup_{k=0}^{\infty} \boldsymbol{A}^k\left(\boldsymbol{x}_0 - \frac{\boldsymbol{u}}{1-\boldsymbol{A}}\right) + \frac{\boldsymbol{u}}{1-\boldsymbol{A}} = [1, 2]$. □

4.2 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 $G = \{\boldsymbol{x} \mid \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}\}$, 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 $\boldsymbol{g}_i = G_i^{\intercal}$. The 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):

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

Applying this to Equation (4.14) we obtain:

$$\rho_{X_n}(\boldsymbol{g}_i) = \rho_{X_0}(\boldsymbol{A}^{\underline{n_i}\intercal}\boldsymbol{g}_i) + \rho_{U^n_{cb}}(\boldsymbol{g}_i) \leq \gamma_i, \tag{4.18}$$

$$\rho_{X_n}(-\boldsymbol{g}_i) = \rho_{X_0}(-\boldsymbol{A}^{\overline{n_i}\intercal}\boldsymbol{g}_i) + \rho_{U^n_{cb}}(-\boldsymbol{g}_i) \leq -\gamma_i. \tag{4.19}$$

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 (2.1), 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

$$\hat{X}^{\sharp}_{\underline{n}} = \mathcal{A}_{\underline{n}} X_0 \oplus \mathcal{B}_{\underline{n}} U \quad \text{(n-reach tube)}$$

$$X^{\sharp}_{\underline{n}} = \boldsymbol{A}^{\underline{n}} X_0 \oplus \mathcal{B}_{\underline{n}} U \quad \text{(n-reach set)}$$

$$\hat{X}^{\sharp}_{\overline{n} \mid \underline{n}} = \tau\left(\mathcal{A}_{\overline{n}-\underline{n}-1} X^{\sharp}_{\underline{n}} \oplus \mathcal{B}_{\underline{n}} U \cap G, U\right)$$

$$\hat{X}^{\sharp}_{\overline{n}} = \hat{X}^{\sharp}_{\overline{n} \mid \underline{n}} \cup \hat{X}^{\sharp}_{\underline{n}}.$$

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

Computing the maximum $\underline{n_i}$ such that Equation (4.18) 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

$$\boldsymbol{g}_i = \sum_{j=1}^{p} k_{ij} \boldsymbol{v}_j + res(\boldsymbol{g}_i), \tag{4.20}$$

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 $\boldsymbol{g}_i$ that lies outside the range of $\boldsymbol{S}$. 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:

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

4.2.1 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

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

Let $\Lambda_\sigma = \{\lambda_i \mid i \in [1, \dots, p],\ \bigwedge_{j=1}^{i-1}(\lambda_i^* \neq \lambda_j \wedge \lambda_i \neq \lambda_j)\}$, be the set of eigenvalues with distinct absolute values (thus 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 $\boldsymbol{J} = diag\,(\boldsymbol{J}_s, s \in [1, \ldots, p_b])$, we have

$$\boldsymbol{J}_\sigma = \begin{bmatrix} \overline{\sigma}_1 & 0 & \cdots & 0 \\ 0 & \overline{\sigma}_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \overline{\sigma}_r \end{bmatrix}, \tag{4.21}$$

where $\overline{\sigma}_s = \sup_{\boldsymbol{x} \neq 0} \frac{\|\boldsymbol{J}_s \boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2}$ is the maximum singular value [120] of the Jordan block $\boldsymbol{J}_s$.

Finally, let

$$\begin{aligned} \boldsymbol{x}'_c &= \frac{1}{2}(\rho_{X'_0}(\boldsymbol{v_i}) + \rho_{X'_0}(-\boldsymbol{v_i})), \quad \boldsymbol{v}_{ij} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}, \\ X'_\sigma &= \{\boldsymbol{x} \mid F_\sigma(\boldsymbol{C}_{X_0}\boldsymbol{S}) f_\sigma(\boldsymbol{x}) \leq f_\sigma(\boldsymbol{d}_{X_0} - \boldsymbol{C}_{X_0}\boldsymbol{S}\boldsymbol{x}'_c)\}, \\ X'_0 &\subseteq f_\sigma^{-1}(X'_{c\sigma}), \text{ where } X'_{c\sigma} = \{f_\sigma(\boldsymbol{x}'_c)\} \oplus X'_\sigma \end{aligned} \tag{4.22}$$

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 centred 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 $j \in [1, \ldots, p_b]$, and therefore

$$\rho_{X_0}((\boldsymbol{A}^n)^\intercal \boldsymbol{g}_i) \leq \sum_{j=1}^{p_b} k_{ij} \overline{\sigma}_j^n \rho_{X_{c\sigma}}((\boldsymbol{v}_\sigma)_j), \tag{4.23}$$

where k_{ij} are the coefficients in Equation (4.20).

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:

$$\rho_{X_0}((\boldsymbol{A}^{\underline{n}_i})^\intercal \boldsymbol{g}_i) \leq \sum_{j=1}^{p_b} k_{ij} \overline{\sigma}_j^{\,\underline{n}_i} \rho_{X_{c\sigma}}((\boldsymbol{v}_\sigma)_j) \leq \gamma_i. \tag{4.24}$$

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

let us define

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

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 (4.24), we obtain

$$\overline{\sigma}^{\,n} \sum_{j=1}^{p} \overline{k}_{ij} \left(\frac{\overline{\sigma}_j}{\overline{\sigma}}\right)^n \leq \gamma_i\ , \tag{4.25}$$

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*

$$\underline{n_i} \geq n = \log_{\overline{\sigma}}(\gamma_i) - \log_{\overline{\sigma}}\left(\sum_{j=1}^{p_b} \overline{k}_{ij} \left(\frac{\overline{\sigma}_j}{\overline{\sigma}}\right)^{\underline{n_i}}\right), \tag{4.26}$$

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

Proof. Notice that the sequence $\underline{n_i}$ is monotonically increasing, before it meets 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 (4.10)) 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

$$\begin{bmatrix} \boldsymbol{x}_k \\ \mathbf{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{Ac} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{k-1} \\ \mathbf{1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ \mathbf{0} \end{bmatrix} \boldsymbol{u}_k,$$

where

$$\begin{bmatrix} \boldsymbol{x}_{k-1} \\ \mathbf{1} \end{bmatrix} \in \left\{ \begin{bmatrix} \boldsymbol{x} \\ \mathbf{1} \end{bmatrix} | \begin{bmatrix} \boldsymbol{G} & \boldsymbol{Gc} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{k-1} \\ \mathbf{1} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{h} \\ \mathbf{1} \end{bmatrix} \right\}.$$

□

4.2.2 Underestimating the number of iterations of a model without inputs

In order to apply a similar technique to (4.19), we must find an equivalent under-approximation. In the case of Equation (4.24), 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 $\lambda_{b_j} = |\lambda_j|$ represents these slower dynamics,

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

which reduces to

$$\overline{\sigma}^{\,n} \sum_{j=1}^{p} \underline{k}_{ij}^{-} \left(\frac{\lambda_{b_j}}{\overline{\sigma}} \right)^n + \overline{\sigma}^{\,n} \sum_{j=1}^{p} \underline{k}_{ij}^{+} \leq -\gamma_i \,, \tag{4.27}$$

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 $\underline{\theta} = \min\{\theta_j \mid j \in [1, \dots, p]\}$, 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:

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

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

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

The solution to the last equation is

$$n = \overline{n}_i - \frac{\sum_{j=1}^{p} c_{ij}\alpha_{ij}\theta_j}{2\sum_{j=1}^{p} c_{ij}\theta_j^2}, \text{ with } n \in [\overline{n}_i, \overline{n}_i + n_\theta]. \tag{4.28}$$

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*

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

$$\overline{n_i} = \overline{n_i}' + k, \text{ given } \rho_{X_0}\left((\boldsymbol{A}^{(\overline{n_i}'+k)})^\intercal \boldsymbol{g}_i\right) \geq \gamma_i, \tag{4.29}$$

where k is the result of Equation (4.28)*. 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.

4.2.3 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 (4.22) 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

$$(X'_{c\sigma})_n = \boldsymbol{J}_\sigma^n X'_{c\sigma} \oplus (\boldsymbol{I} - \boldsymbol{J}_\sigma^n) U_{J\sigma'},$$

$$\rho_{(X'_{c\sigma})_n}(\boldsymbol{v}) = \rho_{X'_{c\sigma}}\left(\boldsymbol{J}_\sigma^{nT}\boldsymbol{v}\right) + \rho_{U'_{J\sigma}}(\boldsymbol{v}) - \rho_{U'_{J\sigma}}\left(\boldsymbol{J}_\sigma^{nT}\boldsymbol{v}\right). \tag{4.30}$$

[3] This is a tighter value than that in [42], where we over-approximated using $n_\theta = \frac{(2\pi)^m}{\prod_j \theta_j}$, where m is the number of conjugate pairs.

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

$$\rho_{((XU)'_\sigma)_n}(\boldsymbol{v}) = \rho_{(XU)'_\sigma}\left(\boldsymbol{J}_\sigma^{nT}\boldsymbol{v}\right), \tag{4.31}$$

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.

4.2.4 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 (4.26). 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 (4.29):

$$\underline{n_i} \geq n_k = \log_{\overline{\sigma}}(\gamma_i) - \log_{\overline{\sigma}}\left(\overline{k}_{\underline{n_i}} + \underline{k}_{\underline{n_k}}\right), \tag{4.32}$$

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 (4.26). 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 (4.32) is smaller than our n in Equation (4.26) ($n_k \leq n \leq \underline{n_i}$ and $\underline{n_i} \geq \underline{n_k}$).

In the case of Equation (4.29), the explicit evaluation of the guard at each cycle executes the behaviour described here.

4.2.5 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 (4.12)).

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 $\{\boldsymbol{v}_{s,i}, i \in [1, \dots, p_s]\}$,*

$$\forall n \geq 0, \boldsymbol{A}^n \boldsymbol{v}_s^i = \lambda_s^n \boldsymbol{v}_{s,i} + \sum_{j=1}^{i-1} \binom{n}{j} \lambda_s^{n-j} \boldsymbol{v}_{s,i-j} = \lambda_s^n \left(\boldsymbol{v}_{s,i} + \sum_{j=1}^{i-1} \binom{n}{j} \lambda_s^{-j} \boldsymbol{v}_{s,i-j} \right). \tag{4.33}$$

Proof. By definition, given an eigenvector $\boldsymbol{v}_s$ of $\boldsymbol{A}$, then $\boldsymbol{A}\boldsymbol{v}_s = \lambda_s \boldsymbol{v}_s$ [103]. Similarly a generalised eigenvector $\boldsymbol{v}_{s,i}$ of $\boldsymbol{A}$ satisfies the equation $(\boldsymbol{A} - \lambda_s \boldsymbol{I})\, \boldsymbol{v}_{s,i} = \boldsymbol{v}_{s,i-1}$ 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:

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

□

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

$$\begin{aligned}\rho_{X_0'}(\boldsymbol{J}^{n\intercal}\boldsymbol{g}_i') &\leq \sum_{j=1}^{p_b} k_{ij}\rho_{X_0}\left(\boldsymbol{J}_b^{n\intercal}f_b(\boldsymbol{v}_j)\right) \\ &\leq \sum_{j=1}^{p_b} k_{ij}\lambda_{bj}^n\rho_{X_0}\left(\boldsymbol{v}_{bs,i'} + \sum_{k=1}^{i'-1}\binom{n}{k}\lambda_{bj}^{-k}\boldsymbol{v}_{bs,i'-k}\right) \\ &\leq \sum_{j=1}^{p_b} k_{ij}\lambda_{bj}^n\left(\rho_{X_0}\left(\boldsymbol{v}_{bs,i'}\right) + \sum_{k=1}^{i'-1}\binom{n}{k}\lambda_{bj}^{-k}\rho_{X_0}\left(\boldsymbol{v}_{bs,i'-k}\right)\right) \\ &\leq \sum_{j=1}^{p_b} k'_{ij0}\lambda_{bj}^n + \sum_{m=1}^{i'} k'_{ijm}\lambda_{bj}^n\prod_{m=0}^{p_s-i'-1}(n-m). \end{aligned} \tag{4.34}$$

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 (4.26) and $k_{ij} = k'_{ij_0} + k'_{ij_m}$, and then do a second iteration using $k_{ij} = k'_{ij_0} + k'_{ij_m}\prod_{m=0}^{p_s-i'-1}(\underline{n}_i' - m)$, which ensures the true value is under the approximation. In the case of $\overline{n}_i$, we also start with $k_{ij} = k'_{ij_0} + k'_{ij_m}$ and update it during the iterative process.

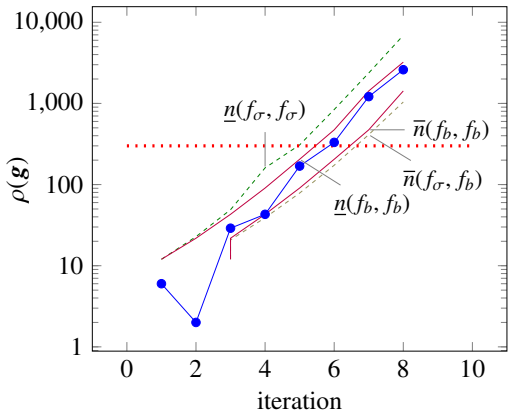

Figure 4.8: Progression of the support function of a system for a given guard. The thick blue dots are real values. The dashed green line over-approximates the progression using singular values (sec 4.2.1), the dashed yellow line under-approximates them using eigenvalue norms (sec 4.2.2), whereas the continuous purple lines represent the tighter over-approximation maintaining the geometric multiplicity (sec 4.2.5). 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.

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

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

with $\boldsymbol{J}_\sigma$ *calculated as in Equation* (4.21)*, initial condition* $\boldsymbol{x}'_0 = [\,1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1\,]$ *and guard set* $\boldsymbol{G}\boldsymbol{x} \leq 300$ *where*

$$\boldsymbol{G} = [\,1 \;\; 3 \;\; -3 \;\; 2 \;\; 4 \;\; 1\,] = [\,1 \;\; 1 \;\; 1 \;\; 2 \;\; 4 \;\; -3\,]\boldsymbol{S}^\intercal.$$

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 4.8

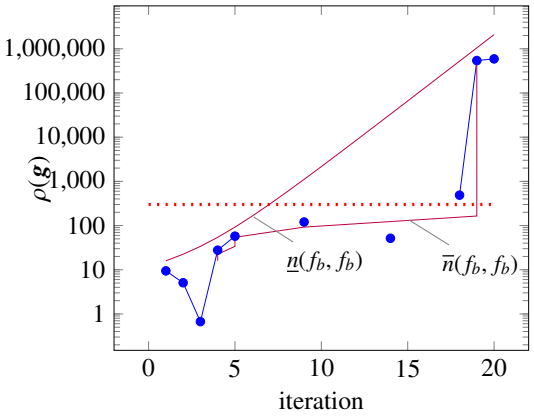

Figure 4.9: Progression of the support function of a rotational system for a given guard. The thick blue dots are real values (negative values are missing due to the log scale). Continuous purple lines represent the over-approximation. The steep vertical line at iteration 19 is due to the alignment of the rotations with the guard at this point. The point at iteration 14 appears below the $\overline{n}$ line because of the alignment of the rotations is in a direction opposite to the guard (as are all points from 10-17). When the initial procedure results in an iteration in this range, the rotation alignment will push the number forward to iteration 19.

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 4.9. 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.

4.2.6 Case study

We have selected a known benchmark from the literature to illustrate the discussed procedure: the room temperature control problem [75]. 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`

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: }
```

function to represent non-deterministic values between 0 and the maximum 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 $G = \left\{ \begin{bmatrix} temp \\ heat \end{bmatrix} \mid \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} temp \\ heat \end{bmatrix} < \begin{bmatrix} 400 \\ 300 \end{bmatrix} \right\}$.

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 eigen-decomposition of the dynamics is (for brevity, the

values are rounded to three decimal places):

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

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 (2.32) to find that the corresponding complex pair remains within the following boundaries:

$$
\mathcal{A}^{32} = \begin{bmatrix} r & i \\ -i & r \end{bmatrix} \begin{Bmatrix} 0.414 < r < 0.985 \\ 0.069 < i < 0.765 \\ 0.108 < r+i < 1.247 \\ 0.916 < i-r < 0.939 \end{Bmatrix}, \; \mathcal{B}^{32} = \begin{bmatrix} r & i \\ -i & r \end{bmatrix} \begin{Bmatrix} 1 < r < 13.41 \\ 0 < i < 17.98 \\ 1 < r+i < 29.44 \\ 6.14 < i-r < 6.52 \end{Bmatrix}.
$$

The reach tube is calculated by multiplying these abstract matrices with the initial sets of states and inputs, as described in Equation (4.15), by the following inequalities:

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

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 4.10, 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 [83] using the given directions. Figure 4.10 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 [83]), and finally the abstractly accelerated set in light yellow (dashed lines). The outer lines represent the guards for G.

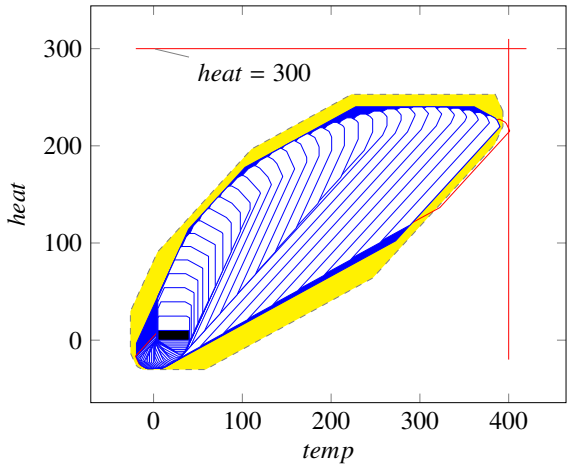

Figure 4.10: The abstractly accelerated tube (yellow, dashed boundary), representing an over-approximation of the thermostat reach tube (dark blue). The set of initial conditions is shown in black, whereas successive reach sets are shown in white. The guards and the reach set that crosses them are close to the boundary in red.

4.3 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 [52].

4.3.1 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 (since we are using pseudo-eigenspaces, 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:

$$\rho_{\lambda_s}(\boldsymbol{v}) = \sum_{j=0}^{p_s} \binom{n}{j} \lambda^{n-j} \boldsymbol{v}_{sj}. \tag{4.35}$$

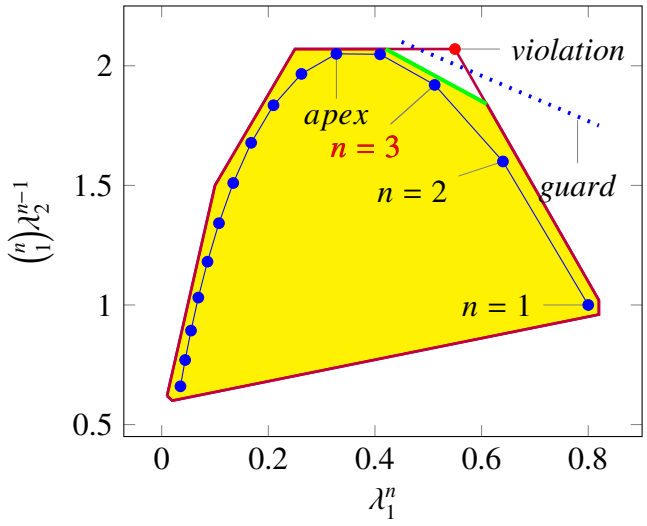

Figure 4.11: Polyhedral faces from an $\mathbb{R}^2$ Jordan block subspace $(\lambda_1^n, \binom{n}{1}\lambda_2^{n-1})$ where $\lambda_1 = 0.8, \lambda_2 = 0.8$, and $1 \leq n \leq 15$. The red dot specifies an abstract vertex violating the safety specification (dashed blue line). The closest iteration 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).

In this case we must use an iterative approximation, as described in Section 4.2.5, to find the closest iteration to the unsafe guard. Although this process is more costly than the ones described above, it is also more 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 4.11. 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.

4.3.2 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 $\mathcal{A}$.

4.3.3 Case study

We have taken an industrial benchmark 'CAFF problem Instance: Building'[4]. 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

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

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 4.1: 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.

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[5] 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 6.1. 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 optimisations (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.

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

4.4 Abstract acceleration of continuous time models

Thus far we have discussed the abstract acceleration of discrete time models. However, on many occasions, these models derive from the discretisation of continuous time models. We therefore seek to establish a sound over-approximation which does not only encompass the selected discretisation, but any chosen discretisation in the time domain.

4.4.1 Acceleration of continuous time models

Accelerated dynamics for continuous time models without inputs

Given a specific time T, the state $\boldsymbol{x}(t = T)$ can be expressed as a linear function of $\boldsymbol{x}_0$ as shown below. We assume that the system has no inputs, so its dynamics are described by $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t)$.

Lemma 4. *The solution to the differential equation $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t)$, where $\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}$, and $\boldsymbol{S}$ are the generalised eigenvectors of $\boldsymbol{A}$, evaluated at time T is*

$$\boldsymbol{x}_T = \boldsymbol{x}(t = T) = \boldsymbol{A}_T\boldsymbol{x}(0) \tag{4.36}$$

$$\boldsymbol{A}_T = \boldsymbol{S}\begin{bmatrix} e^{T\lambda_1} & s_1\frac{T^1 e^{T\lambda_i}}{(2)!} & \cdots & s_i\frac{T^{p-1}e^{T\lambda_i}}{(p-i)!} \\ 0 & e^{T\lambda_i} & s_i\frac{T^{j-i}e^{T\lambda_i}}{(j-i)!} & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & e^{T\lambda_i} \end{bmatrix}\boldsymbol{S}^{-1} \tag{4.37}$$

$$\textit{where } s_i = \begin{cases} 1 & gm(\lambda_i) > 1 \\ 0 & gm(\lambda_i) = 1 \end{cases},$$

$\lambda_i \in \boldsymbol{J}$ are the eigenvalues of $\boldsymbol{A}$, and $gm(\lambda_i)$ is the geometric multiplicity of λ_i. $\boldsymbol{x}_T$ is the state of the system $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t)$ at time t=T. If $\boldsymbol{J}$ is diagonal and there exists a discrete dynamics matrix for a sampling time T_s such that $A_d = e^{\boldsymbol{A}T_s}$, then $\boldsymbol{x}_T = \boldsymbol{A}_d^{\frac{T}{T_s}}\boldsymbol{x}_0$ with $T \in \mathbb{R}_0^{+}$[6] is the state of the system at time $t = T$.

[6]The notation $\boldsymbol{A}_d^{\frac{T}{T_s}}$, corresponds to evaluating a real power of matrix $\boldsymbol{A}_d$ which is not defined in the literature. We use this fractional representation in order to relate the result back to the original matrix $\boldsymbol{A}$ as described in the proof.

Proof. Let us recall Equation 2.11. The discrete representation of the system dynamics discretised with sample time T_1 is ruled by the formula $\boldsymbol{A}_1 = e^{\boldsymbol{A}T_1}$. From matrix theory, we have

$$e^{\boldsymbol{A}} = \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{A}^k,$$

$$e^{\boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}} = \sum_{k=0}^{\infty} \frac{1}{k!} \left(\boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1}\right)^k = \boldsymbol{S}\left(\sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{J}^k\right)\boldsymbol{S}^{-1} = \boldsymbol{S}e^{\boldsymbol{J}}\boldsymbol{S}^{-1}$$

$$\boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1} \Rightarrow \boldsymbol{A}_1 = \boldsymbol{S}\boldsymbol{J}_1\boldsymbol{S}^{-1} = \boldsymbol{S}e^{\boldsymbol{J}T_1}\boldsymbol{S}^{-1}.$$

Let $\boldsymbol{J}$ be a diagonal matrix, such that

$$\lambda_{1i} = e^{\lambda_i T_1}, \forall \lambda_i \in \boldsymbol{J}.$$

Let us now take a different sample rate T_2, such that

$$\lambda_{2i} = e^{\lambda_i T_2}, \forall \lambda_i \in \boldsymbol{J}.$$

We can then say that

$$\lambda_{2i} = e^{\lambda_i T_1 \frac{T_2}{T_1}} = {\lambda_1}_i^{\frac{T_2}{T_1}} \Rightarrow A_2 = A_1^{\frac{T_2}{T_1}}.$$

This proves the theorem for $gm(\lambda_i) = 1$. We now note that given an exponentiated Jordan form with geometric multiplicity, the upper diagonal terms can be referenced to the original eigenvalues modified by the sampling rate. Let $\boldsymbol{J} \in \mathbb{R}^{p \times p}$ be a Jordan Canonical

matrix, then

$$\boldsymbol{J}_1 = \sum_{k=0}^{\infty} \frac{1}{k!} (\boldsymbol{J}T_1)^k = \sum_{k=0}^{\infty} \frac{1}{k!} \begin{bmatrix} \lambda_i^k & \binom{k}{1}\lambda^{k-1} & \dots & \binom{k}{p-1}\lambda_i^{k-p+1} \\ & \lambda_i^k & \binom{k}{1}\lambda_i^{k-1} & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ & & & \lambda_i^k \end{bmatrix} T_1^k.$$

The eigenvalues remain the same as in the diagonal case, but the upper triangular terms are of the form:

$$\begin{aligned} \forall j > i.\, c_{ij} &= \sum_{k=j-i}^{\infty} \frac{1}{k!} \binom{k}{j-i} \lambda_i^{k-(j-i)} T_1^k \\ &= \frac{1}{(j-i)!} \sum_{k=j-i}^{\infty} \frac{1}{(k-(j-i))!} \lambda_i^{k-(j-i)} T_1^k \\ &= \frac{T_1^{j-i} e^{\lambda_i T_1}}{(j-i)!}, \end{aligned}$$

which completes the proof for $gm(\lambda_i) > 1$. □

Accelerated dynamics for continuous time models with parametric inputs

Next, we are interested in studying models with parametric inputs, which are described by $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$, where $\forall t \geq 0.\, \boldsymbol{u}(t) = \boldsymbol{u}$. For such models, a state $\boldsymbol{x}_T$ is computed next.

Lemma 5. *The state of the model $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$ at time T given a parametric input $\boldsymbol{u}$ is*

$$\boldsymbol{x}_T = \boldsymbol{x}(t = T) = \boldsymbol{A}_T \boldsymbol{x}_0 + \boldsymbol{B}_T \boldsymbol{u} \text{ where } \boldsymbol{B}_T = \boldsymbol{A}^{-1}(\boldsymbol{A}_T - \boldsymbol{I})\boldsymbol{B} \tag{4.38}$$

where $\forall t \leq T.\, \boldsymbol{u}(t) = \boldsymbol{u}$, and $\boldsymbol{A}_T$ as in (4.37) [7]*. If $\boldsymbol{J}$ is diagonal and given a sampling time T_s such that $A_d = e^{\boldsymbol{A}T_s}$, then $\boldsymbol{x}_T = A_d^{\frac{T}{T_s}} \boldsymbol{x}_0 + \boldsymbol{A}^{-1}(\boldsymbol{A}_d^{\frac{T}{T_s}} - \boldsymbol{I})\boldsymbol{B}_d \boldsymbol{u}$ with $T \in \mathbb{R}_0^+$ (where $\boldsymbol{A}_d^{\frac{T}{T_s}} = e^{log(\boldsymbol{A}_d)\frac{T}{T_s}} = e^{\boldsymbol{A}T}$) is the state of the system at time T. Note that rather than*

[7] When $\boldsymbol{A}$ is not invertible, we use $\begin{bmatrix} \boldsymbol{A}_T & \boldsymbol{B}_T \\ 0 & \boldsymbol{I} \end{bmatrix} = e^{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ 0 & 0 \end{bmatrix}_t}$.

evauate the function $log(\cdot) = f^{-1}(\cdot)$ *where* $f(\boldsymbol{A}) = e^{\boldsymbol{A}}$*, we use the source directly, i.e.* $log(\boldsymbol{A}_d) = \boldsymbol{A}T_s \Leftrightarrow \boldsymbol{A}_d = e^{\boldsymbol{A}T_s}$

Proof. The first part of the equation ($\boldsymbol{A}_T\boldsymbol{x}_0$) is proven in the previous section. As for the second part, we have from Equation (2.11) that $\boldsymbol{B}_T = \int_{t=0}^{T} e^{\boldsymbol{A}t}dt\ \boldsymbol{B}$ which expands to $\boldsymbol{B}_T = (\boldsymbol{A}^{-1}e^{\boldsymbol{A}T} - \boldsymbol{A}^{-1}e^{\boldsymbol{A}0})\boldsymbol{B} = \boldsymbol{A}^{-1}(\boldsymbol{A}_T - \boldsymbol{I})\boldsymbol{B}$, thus proving the theorem. □

Accelerated dynamics for continuous time models with time varying inputs

Theorem 4. *Let* $U_c = \boldsymbol{S}\boldsymbol{u}'_c$ *be the centre of the eigenspace interval hull of* $\boldsymbol{B}U$ *as described in Section 4.1.5, and* $U_d = \boldsymbol{B}U - U_c$*. The expression*

$$X_T^{\sharp} = \boldsymbol{A}^{-1}X'_T \supseteq X_T \text{ where} \tag{4.39}$$

$$X'_T = \boldsymbol{A}\boldsymbol{A}_T X_0 \oplus F_b^*((\boldsymbol{A}_T - \boldsymbol{I}), U_d) \oplus (\boldsymbol{A}_T - \boldsymbol{I})U_c,$$

is an over-approximation of the reach set at time $t = T$ *of the model* $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$ *given* $\boldsymbol{x}(0) \in X_0$ *and* $\boldsymbol{u}(t) \in U$*.*

Corollary 1. *If* $\boldsymbol{J}$ *is diagonal and there exists a discrete dynamics matrix for a sampling time* T_s *such that* $A_d = e^{\boldsymbol{A}T_s}$*, then* $\forall k \in \mathbb{R}^+$*:*

$$X_k \subseteq \boldsymbol{A}_d^k X_0 \oplus F_b^*\left((\boldsymbol{A}_d^k - \boldsymbol{I})F_b^*\left((\boldsymbol{I} - \boldsymbol{A}_d)^{-1}, U_d\right)\right)$$

$$\oplus\, (\boldsymbol{A}_d^k - \boldsymbol{I})(\boldsymbol{I} - \boldsymbol{A}_d)^{-1}U_c \tag{4.40}$$

is the state of the model at time kT_s*, where* $\boldsymbol{A}_{bd}$ *is a semi-spherical over-approximation of* $\boldsymbol{A}_d$ *as described in Equation* (2.11)*.*

Proof. Let $U'_J = \boldsymbol{S}^{-1}\boldsymbol{B}U$, with $\boldsymbol{u}'_c$ its geometrical centre and U'_b the smallest semi-sphere centred at $\boldsymbol{u}'_c$ containing all points in U'_J (see Equation (4.11)). Then

$$U'_J \subseteq U'_b \oplus \boldsymbol{u}'_c \Rightarrow (\boldsymbol{A}_T - \boldsymbol{I})U' \subseteq (\boldsymbol{A}_T - \boldsymbol{I})U'_b \oplus (\boldsymbol{A}_T - \boldsymbol{I})\boldsymbol{u}'_c.$$

Since $(\boldsymbol{A}_{bT} - \boldsymbol{I})U'_b \supseteq (\boldsymbol{A}_T - \boldsymbol{I})U'_b$, then

$$(\boldsymbol{A}_T - \boldsymbol{I})U' \subseteq (\boldsymbol{A}_{bT} - \boldsymbol{I})U'_b \oplus (\boldsymbol{A}_T - \boldsymbol{I})\boldsymbol{u}'_c,$$

thus proving the theorem for $\boldsymbol{u}(t) = \boldsymbol{u}_0$ with $t \in [0, T]$.

For the continuous time interval, let us look at the solution for the differential equation at time T with a non-deterministic input $\boldsymbol{u}$.

$$\boldsymbol{x}_T = e^{\boldsymbol{A}T}\boldsymbol{x}_0 + \int_{t=0}^{T_s} e^{\boldsymbol{A}t}\boldsymbol{B}\boldsymbol{u}(t)dt.$$

We will first divide this equation into

$$\begin{aligned}\boldsymbol{x}_T &= e^{\boldsymbol{A}T}\boldsymbol{x}_0 + \int_{t=0}^{T} e^{\boldsymbol{A}t}\boldsymbol{B}\boldsymbol{u}_c dt + \int_{t=0}^{T_s} e^{\boldsymbol{A}t}\boldsymbol{B}(\boldsymbol{u}(t) - \boldsymbol{u}_c)dt \\ &= \boldsymbol{A}_T\boldsymbol{x}_0 + \boldsymbol{B}_T\boldsymbol{u}_c + \int_{t=0}^{T}\sum_{k=0}^{\infty}\frac{1}{k!}\boldsymbol{A}^k t^k \boldsymbol{B}(\boldsymbol{u}(t) - \boldsymbol{u}_c)dt.\end{aligned}$$

It follows from Equation (4.11) that $\boldsymbol{A}^k\boldsymbol{B}(\boldsymbol{u}(t) - \boldsymbol{u}_c) \subseteq F_b^*(\boldsymbol{A}^k, \boldsymbol{B}(U - \boldsymbol{u}_c))$, hence

$$\begin{aligned}X_T \subseteq &\boldsymbol{A}_T X_0 + \boldsymbol{B}_T\boldsymbol{u}_c \\ &+ \int_{t=0}^{T}\sum_{k=0}^{\infty}\frac{1}{k!}t^k F_b^*\left(\boldsymbol{A}^k, \boldsymbol{B}(U - \boldsymbol{u}_c)\right)dt \\ \subseteq &\boldsymbol{A}_T X_0 + \boldsymbol{B}_T\boldsymbol{u}_c + \int_{t=0}^{T_s} F_b^*\left(e^{\boldsymbol{A}t}, \boldsymbol{B}(U - \boldsymbol{u}_c)\right)dt \\ \subseteq &\boldsymbol{A}_T X_0 + \boldsymbol{B}_T\boldsymbol{u}_c + \boldsymbol{A}^{-1}F_b^*\left((\boldsymbol{A}_T - \boldsymbol{I}), \boldsymbol{B}(U - \boldsymbol{u}_c)\right).\end{aligned}$$

Multiplying by $\boldsymbol{A}$ we get X'_T as in (4.39), thus proving the theorem. □

4.4.2 Finding abstract supports for continuous dynamics

In the case of discrete dynamics, support functions can be found by evaluating hyperplanes of consecutive exponents of the eigenvalues using differences. In the case of continuous time, this translates to calculating derivatives. Once again, when the sets are known to be convex, we may calculate the normal vector to this derivative and evaluate its support function on the selected iteration. The direction of the vector is determined by evaluating the support function of another point in the curve, which is known to be smaller than that of the selected point. The corresponding derivatives are:

1. Positive real eigenvalues

   We first calculate the slope of the tangent between the two progressions: $\frac{df(k)}{dg(k)} = \frac{f'(k)}{g'(k)}$. Hence $\frac{d\lambda_1^k}{d\lambda_2^k} = \frac{log(\lambda_1)\lambda_1^k}{log(\lambda_2)\lambda_2^k}$. From this we know that the polar angle of the support vector is $\psi = \tan^{-1}(\frac{log(\lambda_1)\lambda_1^k}{log(\lambda_2)\lambda_2^k}) + \frac{\pi}{2}$

2. Complex conjugate eigenvalue pairs

   The pair forms a logarithmic spiral. The tangent of a spiral with polar equation $r = ae^{b\theta}$ will have a derivative $\frac{dr}{d\theta} = bae^{b\theta} = br$, hence the angle of the vector normal to the curve is $\tan^{-1}(\frac{r}{\frac{dr}{d\theta}}) + \frac{\pi}{2} = \tan^{-1}(\frac{1}{b}) + \frac{\pi}{2}$

3. Equal eigenvalues

   The supports are as on the discrete time case orthogonal to the plane $x = y$ and within the square containing the largest and lowest values for the pair.

4. Negative real eigenvalues and mixed types

   Since we use the norm of the eigenvalues, we compute the tangents as in the case for positive real eigenvalues and apply the mirror image as in the discrete time case.

5. Jordan blocks with geometric multiplicity > 1 In the case of intra-block (i.e. within the same Jordan block) supports, derivation is straight forward since the gradient depends on a power of T and can be done in the same fashion as the corresponding item above. In the case of inter-block supports, the function may have a convergent and a divergent element. If this is the case we find where the apex of this function is (where the gradient is 0) and use this as the inequality. In the strictly divergent

case, we may use the same mechanisms described above. In our implementation we err on the side of caution, using hyper-boxes for the more complex cases of inter-block relations.

4.4.3 Calculating the number of iterations for continuous dynamics

Following the same steps as in Lemma 3 we develop an equivalent for continuous dynamics.

Lemma 6. *Given a matrix $\boldsymbol{A}$ with eigenvalues $\{\lambda_s \mid s \in [1, \dots, r]\}$, where each eigenvalue λ_s has a geometric multiplicity p_s and corresponding generalised eigenvectors $\{\boldsymbol{v}_{s,i} \mid i \in [1, \dots, p_s]\}$,*

$$\forall t \geq 0, \boldsymbol{A}_t \boldsymbol{v}_s^i = e^{\boldsymbol{A}t}\boldsymbol{v}_s = e^{\lambda_s t}\boldsymbol{v}_{s,i} + \sum_{j=1}^{i-1} t^j e^{\lambda_s t}\boldsymbol{v}_{s,i-j} = e^{\lambda_s t}\left(\boldsymbol{v}_{s,i} + \sum_{j=1}^{i-1} t^j \boldsymbol{v}_{s,i-j}\right). \tag{4.41}$$

Proof. The proof derives again from the Taylor expansion.

$$\begin{aligned} e^{\boldsymbol{A}t}\boldsymbol{v}_{s,i} &= \sum_{k=0}^{\infty} \boldsymbol{A}^k \frac{t^k}{k!}\boldsymbol{v}_{s,i} = \sum_{k=0}^{\infty} \frac{t^k}{k!}\left(\lambda_s^k \boldsymbol{v}_{s,i} + k\lambda_s^{k-1}\boldsymbol{v}_{s,i-1}\right) \\ &= \sum_{k=0}^{\infty} \frac{t^k}{k!}\lambda_s^k \boldsymbol{v}_{s,i} + \sum_{k=0}^{\infty} \frac{t^k}{k!} k\lambda_s^{k-1}\boldsymbol{v}_{s,i-1} \\ &= e^{\lambda_i t}(\boldsymbol{v}_{s,i} + t\boldsymbol{v}_{s,i-1}). \end{aligned} \tag{4.42}$$

The rest of the proof follows the same expansion as in 3. □

Given the similarity of Equation (4.41) with (4.33) we may apply exactly the same techniques described in Section 4.2.5 to the continuous case.

name	characteristics type	dim	inputs	bounds	new bounds IProc	Sting	analysis time [sec] 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 [107]; **Sting** is [56]; **mpfr** is Axelerator (this work) using 256 bit mantissas;

Table 4.2: Experimental comparison of unbounded-time analysis tools with inputs

4.5 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. 4.3). 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/`

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 $\pm x_i \leq bound_k$).

4.5.1 Comparison with other unbounded-time approaches

In a first experiment we have benchmarked our implementation against the tools IProc [107] and Sting [56]. We have tested these tools on different discrete time models,

name	characteristics type	dim	bounds	improved tighter	looser	analysis time (sec) 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. [108]; **mpfr+** is this work 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 the Jordan form

Table 4.3: Experimental comparison with previous work

including guarded/unguarded ones, stable/unstable ones, and models with complex/real loops with inputs (details in Table 4.2). 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 4.3 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) [108] (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 4.2 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 [108]).

At the same time, our implementation is faster than [108], partly due to the use of numeric eigen-decomposition (as opposed to symbolic), but mostly in view of the cost of increasingly large rational representations in [108] 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 [108] (using the ATLAS package), whereas our implementation requires at most one second. For the last two benchmarks, the polyhedral computations blow up in [108], whereas our support function approach shows only moderately increasing runtimes. The increase of speed is owed to multiple factors, as detailed in Table 4.4. 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.

4.5.2 Comparison with bounded-time approaches

In a third experiment, we compare our method with the LGG algorithm [96] used by SpaceEx [83]. 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 [96] within our software. We have run both methods on the convoyCar example [108] with inputs, which presents an unguarded, scalable (the dimension can be increased by adding more cars), stable loop with complex dynamics, and we have 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 4.5). However, by

Optimisation	Speed-up
Eigen vs. ATLAS [8]	2–10
Support functions vs. logahedral templates	2–40
long double vs. multiple precision arithmetic	5–200
interval vs. regular arithmetic	.2–.5
Total	4–80000

Table 4.4: Performance improvements over [108]

	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 4.5: Comparison on convoyCar2 benchmark [108], between this work and the LGG algorithm from [96]. The intervals show the minimum and maximum values obtained for each variable.

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 spiraling dynamics. Still, in many applications abstract acceleration can provide a tight over-approximation for the convex hull of (possibly non-convex) reach tubes.

Table 4.5 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 performs better in terms of tightness, however unlike our approach, its runtime distinctly increases with the number of iterations. Our implementation of LGG [96] 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.

4.5.3 Comparison with alternative abstract acceleration techniques

Table 4.6 shows a comparison between our approach and [122]. 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 [122] 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

	Axelerator		Axelerator using [122]	
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 4.6: Comparison on convoyCar2 benchmark [108], between this work and the work in [122]. The intervals show the minimum and maximum values obtained for each variable.

tool	time	sound
Axelerator U	34 s	no
Axelerator S	1123 s	yes
Flow*	165 s	yes
SpaceEx	2.2 s	no
CORA	6.5 s	no

Table 4.7: Comparison on the building benchmark, between this work (in continuous mode) and continuous time BMC algorithms.

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 [122], 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 [122] creates one divergent eigenvalue, which causes a critical change in the behaviour of the dynamics that leads to unbounded results.

4.5.4 Comparison with continuous time approaches

The implementation (Axelerator) was compared with several state of the art tools for Bounded Model Checking of continuous time LTI systems. The results are shown on Table 4.7

Although the performance is significantly slower than these tools, we remark on several aspects that redeem our approach. First of all it is unbounded in time, which means that given a long enough time horizon it should outperform all tools in this set. Secondly, we note that the fastest tool, SpaceEx, took nearly 900 seconds when the benchmark was initially introduced in [161]. This reflects the importance of both the implementation and optimisations that this algorithm has benefited from during the course

of the last years. Thirdly, our implementation leverages heavily on the Simplex algorithm. Our sound simplex is based on a naive implementation without heuristics. We believe that, using such heuristics, processing speeds could easily be improved by a factor of 4, placing our algorithm on par with the ones presented here. Finally, we note that the refinement process for this benchmark results in a larger processing time, which means that for a more relaxed specification our tool is faster than presented here (3 seconds for unsound double precision when the bound is .1), indicating that it could indeed outperform these tools in several large scale benchmarks.

4.5.5 Scalability

Finally, in terms of scalability, we have an expected $O(p^3)$ typical complexity w.r.t. the number of variables p, which derives from the simplex algorithm and the matrix multiplications in Equation (4.15). We have parametrised the number of cars in the convoyCar example [108] (also seen in Table 4.3), 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 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.

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

Chapter 5

Control Synthesis using Abstract Acceleration

Verification of LTI models is a useful tool for controller design. It enables engineers to verify the proposed solutions of a control feedback problem in order to ensure that these solutions are sound. The design of such models is highly complex though, requiring lots of expertise which could lead to many iterations of the design-verification process when the model is not well understood. By comparison, automating the design process alleviates the efforts of the engineer and may present overall better solutions when the synthesis engine can reason about more complex specs than a given engineer is able to. We will now address the problem of stateful digital controller synthesis for LTI physical plants, evaluating properties such as safety, stability and settling time.

We perform digital control synthesis over a hybrid model, where the plant exhibits continuous behaviour whereas the controller operates in discrete time and over a quantised domain. Our model incorporates the effects of the quantisers (A/D and D/A converters) and of time discretisation, as well as representation errors introduced by modelling artefacts such as an observer [19] in a finite-precision domain, while reasoning about high-level properties such as stability, robustness, safety, and response time, which define the behaviour of the model. In particular, safety requirements are

frequently overlooked in conventional feedback control synthesis, but play an important role in systems engineering.

We use a CounterExample-Guided Inductive Synthesis (CEGIS) [109, 154] approach, which we enhance with abstraction refinement and optimisation resulting in the novel CEGIS-OR (CEGIS with Optimisation and Refinement). This design uses an iterative process where each iteration performs inductive generalisation based on counterexamples provided by a verification oracle. Essentially, the inductive generalisation uses information about a limited number of inputs to make claims in the form of candidate solutions about all the possible inputs. By using abstraction refinement, CEGIS-AR allows us to handle complex models, which are initially represented by coarse abstractions that only get refined on demand. Consequently, solutions that do not require a high level of precision are obtained fast.

However, the use of coarse abstractions may result in low-quality candidate solutions (i.e., candidates that fail the specification on a very large number of inputs thus requiring further generalisation). To overcome this problem, we introduce CEGIS-OR, which uses optimisation techniques after each inductive generalisation phase to improve the produced candidates. In particular, our optimisation procedure reduces the size of the reach space of the closed loop model, while maintaining other dynamical properties of interest (safety, stability, and response time).

Let us recall the observer model presented in Section 2.6.4. Figure 5.1 shows a physical plant controlled by a digital observer. Our intention is to find adequate values for the matrices $*_{\langle C \rangle}$ that ensure the sound operation of the closed loop model given a set of user-defined properties such as stability, safety, response time, sampling time and quantisation thresholds.

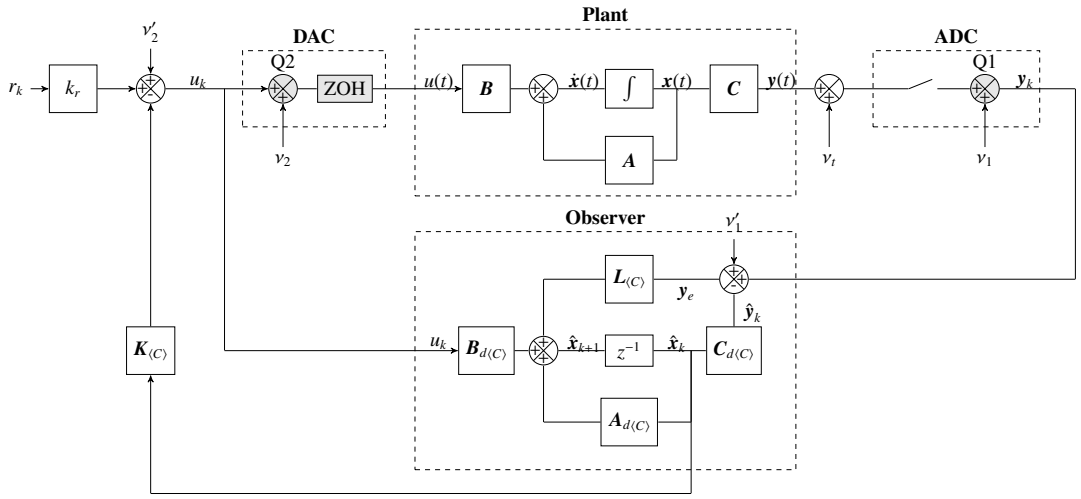

Figure 5.1: Closed-loop digital control system with observer

5.1 Accelerated dynamics for continuous time models with discrete time feedback inputs

Before we progress, we must analyse the case of hybrid time models, where the plant dynamics evolve in continuous time while the feedback dynamics evolve in discrete time. A new input value is produced only at every sample time but implemented over the continuous-time plant dynamics.

Theorem 5. *The expression*

$$\boldsymbol{x}_T = (\boldsymbol{A}_{T-kT_s} - \boldsymbol{B}_{T-kT_s}\boldsymbol{K})(\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K})^k\boldsymbol{x}_0, \tag{5.1}$$

with $\boldsymbol{A}_T, \boldsymbol{B}_T$ *as per Equations* (4.37) *and* (4.38)*, and* $\boldsymbol{A}_d, \boldsymbol{B}_d$ *as per Equation* (2.11) *is the state of the model*

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}(t),$$

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}_k, \text{ where } kT_s \leq t < (k+1)T_s,$$

Proof. Let $\boldsymbol{K}$ be a feedback controller such that $\boldsymbol{u}_k = -\boldsymbol{K}\boldsymbol{x}_k$ at time $t = kT_s$. Let the output

of the feedback system be a zero order holder such that $\boldsymbol{u}(t' + kT_s) = \boldsymbol{u}(kT_s)$, for $0 \leq t' < T_s$. The closed loop dynamics of the model between times kT_s and $kT_s + T_s$ are

$$\boldsymbol{x}_{k,t'} = \boldsymbol{A}_{t'}\boldsymbol{x}_k - \boldsymbol{B}_{t'}\boldsymbol{K}\boldsymbol{x}_k = (\boldsymbol{A}_{t'} - \boldsymbol{B}_{t'}\boldsymbol{K})\boldsymbol{x}_k, \text{ where } 0 \leq t' < T_s.$$

We further explore the dynamics across discrete time boundaries. Let $\boldsymbol{A}_d = \boldsymbol{A}_{T_s}$ and $\boldsymbol{B}_d = \boldsymbol{B}_{T_s}$

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1,T_s} = (\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K})\boldsymbol{x}_{k-1},$$

$$\boldsymbol{x}_{k,T_s} = (\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K})(\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K})\boldsymbol{x}_{k-1},$$

which accelerating results and setting $T = kT_s + t'$ becomes

$$\boldsymbol{x}_k = (\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K})^k\boldsymbol{x}_0, \tag{5.2}$$

$$\boldsymbol{x}_T = (\boldsymbol{A}_{T-kT_s} - \boldsymbol{B}_{T-kT_s}\boldsymbol{K})(\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K})^k\boldsymbol{x}_0. \tag{5.3}$$

□

5.2 Errors generated by numerical representations

5.2.1 Modelling quantisation as noise

In Figures 2.4 and 2.5 we introduced several sources of noise, which are due to quantisation effects. We now describe their nature and create a sound model for them to be used during our synthesis and verification process.

During any given ADC conversion, the continuous signal will be sampled in the real domain and transformed by $\mathcal{F}_{\langle A \rangle}(x)$. This sampling uses a threshold which is defined by the less significant bit ($q_a = 2^{-F_a}$) of the ADC and some non-linearities of the circuitry.

The overall conversion is

$$\boldsymbol{y}_k = \mathcal{F}_{\langle A \rangle}\left(y(kT_s)\right), \text{ with } \boldsymbol{y}_k \in \left[y(kT_s) - \frac{q_a}{2}, y(kT_s) + \frac{q_a}{2}\right].$$

where $t = kT_s$, and T_s is the sampling time and k the current iteration of the loop. If we denote the error in the conversion by $\nu_k = y_k - y(t)$, then we may define some bounds for it $\nu_k \in [-\frac{q_a}{2}, \frac{q_a}{2}]$.

We will assume, for the purposes of this analysis, that the domain of the ADC is that of the digital controller (i.e., the quantiser includes a digital gain added in the code resulting in $\mathbb{R}_{\langle A \rangle} = \mathbb{R}_{\langle C \rangle}$). The process of quantisation in the DAC is similar except that it is calculating $\mathcal{F}_{\langle D \rangle}(\mathcal{F}_{\langle C \rangle}(x))$. If these domains are the same ($\mathbb{R}_{\langle D \rangle} = \mathbb{R}_{\langle C \rangle}$), or if the DAC resolution in higher than the ADCs ($\mathbb{R}_{\langle D \rangle} \supseteq \mathbb{R}_{\langle C \rangle}$), then the DAC quantisation error is equal to zero. From the above equations we can now define the ADC and DAC quantisation noises $\nu_{1k} \in [-\frac{q_a}{2}, \frac{q_a}{2}]$ and $\nu_{2k} \in [-\frac{q_d}{2}, \frac{q_d}{2}]$, where given the above assumptions $q_d = q_a = q_c$. This is illustrated in Figures 2.4 and 2.5 where Q_1 is the quantiser of the ADC and Q_2 the quantiser for the DAC. These bounds hold irrespective of whether the noise is correlated, hence we may use them to over-approximate the noise effect on the state space progression over time. The resulting dynamics are

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_d \boldsymbol{x}_k + \boldsymbol{B}_d(u_k + \nu_{2k}), \quad u_k = -\boldsymbol{K}\boldsymbol{x}_k + \nu_{1k},$$

which result in the following closed-loop dynamics:

$$\boldsymbol{x}_{k+1} = (\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K}_d)\boldsymbol{x}_k + \boldsymbol{B}_d\nu_{2k} + \nu_{1k}\,.$$

Round-off errors

Let $q_* = 2^{-F_*}$ be the smallest difference between two numbers in their corresponding domains. Assuming that our system truncates numbers to the nearest value, this means that the error δ introduced by the function $\mathcal{F}_{\langle * \rangle}$ is $\delta_* = \mathcal{F}_{\langle * \rangle}(x) - x \in \left[-\frac{q_*}{2}, \frac{q_*}{2}\right]$. We note

that in the case of floating point,the choice of F_* is taken from the largest multiplication found (e.g., $\|\boldsymbol{A}\|_1\|\boldsymbol{x}\|_1$, where $\boldsymbol{x} \models \phi_{safety}$, where ϕ_{safety} is defined in 5.8). From these equations we can define:

$$\nu_1 \in \left[-\frac{q_a}{2}, \frac{q_a}{2}\right] \cap \left[-\frac{q_c}{2}, \frac{q_c}{2}\right], \nu_2 \in \left[-\frac{q_d}{2}, \frac{q_d}{2}\right].$$

We may also bound ν_2' as follows:

$$\begin{aligned}
\delta\left(\boldsymbol{K}_{\langle C\rangle}\boldsymbol{x}_{\langle C\rangle}\right) &\in \left[-p\frac{q_c}{2},\ p\frac{q_c}{2}\right] \\
\boldsymbol{\nu_{ei}} &\in \left[-(p+2)\frac{q_c}{2}, (p+2)\frac{q_c}{2}\right], i \in [1, \ldots, p] \\
\Rightarrow \nu_2' &\in \left[-p(\|\boldsymbol{K}_{\langle C\rangle}\|_1 + 2)\frac{q_c}{2},\ p(\|\boldsymbol{K}_{\langle C\rangle}\|_1 + 2)\frac{q_c}{2}\right].
\end{aligned}$$

Similarly we also find

$$\nu_1' \in \left[-p(\|\boldsymbol{C}_{d\langle C\rangle}\|_1 + 2)\frac{q_c}{2},\ p(\|\boldsymbol{C}_{d\langle C\rangle}\|_1 + 2)\frac{q_c}{2}\right].$$

The proof derives from $\|\boldsymbol{K}\boldsymbol{x}\|_1 \leq \|\boldsymbol{K}\|_1\|\boldsymbol{x}\|_1$.

5.2.2 Time delay on discrete time feedback models

A common occurrence in digital systems is that they do not necessarily satisfy real-time specifications, in view of many errors introduced in the dynamics. Errors might result from delays due to processing time, corruptions over communication lines in the control loop, and approximations due to digitalisation. In our model, delays are all deferred to the Zero Order Holder (ZOH) of the DAC (See Figure 5.1). In the digital world everything takes place at times $t = kT_s$ for $k \in \mathbb{N}$ (discrete time progression), and the output signal is delayed at the DAC when fed back to the continuous-time domain at time ($t = kT_s + T_d$).

Theorem 6. *The error introduced by a delay T_d in the control input $\boldsymbol{u}_k$ updated by a discrete feedback controller for a continuous LTI system can be modeled as a non-deterministic disturbance signal (ν_t in Figure 5.1) upper bounded by*

$$\|\boldsymbol{y}_k - \boldsymbol{y}_{T_d k}\|_1 \leq q_t, \text{ where } q_t = (\|\boldsymbol{C}(\boldsymbol{B}_{T_s} - \boldsymbol{B}_{t_s})\|_1 + \|\boldsymbol{C}\boldsymbol{A}_{t_s}\boldsymbol{B}_{T_d}\|_1)\|U\|_1, \tag{5.4}$$

where $\boldsymbol{y}_{T_d}$ is the output of the model with delay T_d, $t_s = T_s - T_d$, and $\|\boldsymbol{x}\|_1 = \sum_{i=0}^{size(\boldsymbol{x})} |\boldsymbol{x}_i|$ is the l_1 norm, which we extend to sets as $\|X\|_1 = \sup\{\|\boldsymbol{x}\|_1 \mid \boldsymbol{x} \in X\}$.

Proof. Let us look at Equations (5.2) and (5.3). When we introduce an arbitrary delay T_d where $t_s = T_s - T_d$ we obtain:

$$\begin{aligned}\boldsymbol{x}_k &= \boldsymbol{A}_{t_s}(\boldsymbol{A}_{T_d}\boldsymbol{x}_{k-1} - \boldsymbol{B}_{T_d}\boldsymbol{K}\boldsymbol{x}_{k-2}) - \boldsymbol{B}\boldsymbol{K}\boldsymbol{x}_{k-1} \\ &= (\boldsymbol{A}_{T_s} - \boldsymbol{B}_{t_s}\boldsymbol{K})\boldsymbol{x}_{k-1} - \boldsymbol{A}_{t_s}\boldsymbol{B}_{T_d}\boldsymbol{K}\boldsymbol{x}_{k-2}.\end{aligned}$$

Differentiating with the expected progression without delay, we get a delay error:

$$\begin{aligned}\boldsymbol{x}_{ek} &= (\boldsymbol{B}_{T_s} - \boldsymbol{B}_{t_s})\boldsymbol{K}\boldsymbol{x}_{k-1} - \boldsymbol{A}_{t_s}\boldsymbol{B}_{T_d}\boldsymbol{K}\boldsymbol{x}_{k-2} \\ &= (\boldsymbol{B}_{T_s} - \boldsymbol{B}_{t_s})\boldsymbol{u}_k - \boldsymbol{A}_{t_s}\boldsymbol{B}_{T_d}\boldsymbol{u}_{k-1}.\end{aligned}$$

Rather than explicitly calculate this error, we will find an upper bound. Let $U = \bigcup \boldsymbol{u}_k$. Propagating the error to the output $\boldsymbol{y}_k$ we have:

$$\begin{aligned}\boldsymbol{y}_{ek} &\subseteq \boldsymbol{C}(\boldsymbol{B}_{T_s} - \boldsymbol{B}_{t_s})U \oplus \boldsymbol{C}\boldsymbol{A}_{t_s}\boldsymbol{B}_{T_d}U \\ \Rightarrow \|\boldsymbol{y}_{ek}\|_1 &\leq (\|\boldsymbol{C}(\boldsymbol{B}_{T_s} - \boldsymbol{B}_{t_s})\|_1 + \|\boldsymbol{C}\boldsymbol{A}_{t_s}\boldsymbol{B}_{T_d}\|_1)\|U\|_1\end{aligned}$$

□

This result allows us to over-approximate the effects of the delay by introducing a

disturbance signal $\nu_t = [-q_t, q_t]$ at the output. This noise will be small when the delay is much smaller than the sampling time ($t_s \approx T_s$ and $\boldsymbol{B}_{T_d} \approx 0$).

5.2.3 Modelling errors

Our synthesis implementation is based on SAT-solving. This means we use bit-vectors to represent the dynamics and the state space. As discussed in Section 5.2, this introduces a number of errors that we need to account for in our process. Errors incurred by the use of numerical algorithms during synthesis are not accounted for but rather checked during the verification phase. If the counterexample provided corresponds to a pre-existing one, the synthesiser then increases its precision to reduce this error.

Previous studies [123] show that the FWL affects the poles and zeros positions, degrading the closed-loop dynamics, causing steady-state errors and eventually de-stabilizing the system [27]. However, since the verifier also checks for the stability of the dynamics as a precursor to safety, this case is handled inside the loop. In the following, we shall disregard these steady-state errors (caused by FWL effects) when stability is ensured by synthesis, and then verify its safety accounting for the finite-precision errors.

5.3 Properties of a controllable model with observer

Since we are interested in synthesizing controllers with observers, in this section we will look at checking stability, safety and the response time in the presence of an observer.

5.3.1 Stability of an observed model with FWL effects

Stability qualitatively denotes the property of a model to evolve within a bounded set around an equilibrium point. In this paper, we are interested in asymptotic stability, where the model is further expected to converge to the steady state value (i.e., the equilibrium point). We specify model stability via Jury's criterion [74], as classical in Control Theory, which is given as a requirement expressed as an algebraic formula on the characteristic

polynomial of the model. In order to facilitate the use of Jury's criterion, we transform the model into a representation that contains the coefficients of its characteristic polynomial expressed as sums of controller and plant values. The benefit of this representation is that it has no multiplications, which are expensive to handle by SAT solvers (used by the synthesiser). Such a representation is a model known as "canonical form" [20] and is dynamically equivalent to the given model. In this section we explore the implications of using the canonical form, alongside the use of FWL representations.

We assume that the domain of the observer model is discretised, i.e., the domain for both the controller and the observer is $\mathbb{R}_{\langle C \rangle}$. This means that we have matrices $\boldsymbol{A}_{d\langle C \rangle}, \boldsymbol{B}_{d\langle C \rangle}, \boldsymbol{C}_{\langle C \rangle}, \boldsymbol{K}_{\langle C \rangle}, \boldsymbol{L}_{\langle C \rangle} \in \mathbb{R}^{*}_{\langle C \rangle}$, where $*$ represents the appropriate dimension for each matrix. New problems arise due to the FWL truncation of these matrices. These are reflected in the canonical form of the resulting model (see $\boldsymbol{A}_e$ and $\boldsymbol{B}_e$), which is expressed by:

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{x}'_e \end{bmatrix}_{k+1} = \begin{bmatrix} \boldsymbol{A}_c + \boldsymbol{B}_e & \boldsymbol{B}_o \\ \boldsymbol{A}_e & \boldsymbol{A}_o \end{bmatrix} \begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{x}'_e \end{bmatrix}_k + \boldsymbol{B}' \begin{bmatrix} \boldsymbol{r}_k + \nu_2 + \nu'_2 \\ \nu_1 + \nu'_1 + \nu_t \end{bmatrix}, \tag{5.5}$$

$$\boldsymbol{A}_c = \begin{bmatrix} k_{p-1} - a_{p-1} & \cdots & k_1 - a_1 & k_0 - a_0 \\ & & & 0 \\ & \boldsymbol{I} & & \vdots \\ & & & 0 \end{bmatrix} \quad \boldsymbol{A}_o = \begin{bmatrix} l_{p-1} - a_{p-1} & & & \\ \vdots & & \boldsymbol{I} & \\ l_1 - a_1 & & & \\ l_0 - a_0 & 0 & \cdots & 0 \end{bmatrix},$$

where $\boldsymbol{x}'$ is the state of the model in canonical form and $\boldsymbol{x}'_e$ is the error of the observer evaluated in the transformed domain; further, $\boldsymbol{A}_c$ the closed-loop matrix of the controller in canonical form, $\boldsymbol{A}_o$ the closed-loop matrix of the observer in canonical form, $\boldsymbol{B}_o$ the a feedback gain matrix, $\boldsymbol{B}'$ the noise sensitivity matrix, and $\boldsymbol{A}_e$ and $\boldsymbol{B}_e$ are the error caused by the FWL truncation, seen after the transformation to canonical form.

Given the matrices $\boldsymbol{T}_c$ and $\boldsymbol{T}_o$ that transform the model into canonical form [20], we

may calculate the above matrices as follows:

$$\boldsymbol{A}_c = \boldsymbol{T}_c \left(\boldsymbol{A}_d - \boldsymbol{B}_d \boldsymbol{K}_{\langle C \rangle}\right) \boldsymbol{T}_c^{-1} \qquad \boldsymbol{A}_o = \boldsymbol{T}_o \left(\boldsymbol{A}_{d\langle C \rangle} - \boldsymbol{L}_{\langle C \rangle} \boldsymbol{C}_{d\langle C \rangle}\right) \boldsymbol{T}_o^{-1}$$

$$\boldsymbol{B}_o = \boldsymbol{T}_c \boldsymbol{B}_{d\langle C \rangle} \boldsymbol{K}_{\langle C \rangle} \boldsymbol{T}_o^{-1} \qquad \boldsymbol{B}_e = \boldsymbol{T}_c (\boldsymbol{B}_d - \boldsymbol{B}_{d\langle C \rangle}) \boldsymbol{K}_{\langle C \rangle} \boldsymbol{T}_o^{-1}$$

$$\boldsymbol{A}_e = \boldsymbol{T}_o \left(\boldsymbol{A}_d - \boldsymbol{A}_{d\langle C \rangle} - \boldsymbol{L}_{\langle C \rangle} \boldsymbol{C}_e\right) \boldsymbol{T}_o^{-1} \qquad \boldsymbol{C}_e = \boldsymbol{C}_d - \boldsymbol{C}_{d\langle C \rangle}$$

$$\boldsymbol{C}_t = \boldsymbol{C}_d \boldsymbol{T}_c$$

$$\boldsymbol{A}' = \begin{bmatrix} \boldsymbol{A}_c + \boldsymbol{B}_e & \boldsymbol{B}_o \\ \boldsymbol{A}_e & \boldsymbol{A}_o \end{bmatrix} \qquad \boldsymbol{B}' = \begin{bmatrix} \boldsymbol{T}_c \boldsymbol{B}_{d\langle C \rangle} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{T}_o \boldsymbol{L}_{\langle C \rangle} \end{bmatrix}$$

Moreover, ν_1 and ν_2 are the quantisation noises introduced by the ADC and the DAC, ν_t is the noise introduced by the delay, and ν_1' and ν_2' are quantisation noises caused by round-offs in the FWL operations.

In a real domain $\mathbb{R}$, the characteristic polynomial of the closed loop dynamics is defined by the characteristic polynomials of $\boldsymbol{A}_c$ and $\boldsymbol{A}_o$ as

$$P(z) = P_K(z) P_L(z) = \left(z^p + \sum_{i=0}^{p-1} (a_i - k_i) z^i\right) \left(z^p + \sum_{i=0}^{p-1} (a_i - l_i) z^i\right).$$

Jury's criterion [74] uses these coefficients to ensure that no root of the polynomial has a magnitude greater than one. The introduction of the FWL errors causes this to no longer be true. We must therefore find a suitable solution to maintain this independence of polynomials and use the given coefficients by bounding their error w.r.t. the true coefficients of the closed loop matrix.

Gershgorin disks [162] allow us to limit the error in the expected eigenvalues. Let $\delta_o = \max(\|\boldsymbol{A}_{e_{i,*}}\|_1)$ (i.e., the largest row norm in $\boldsymbol{A}_e$) be the largest error bound for any eigenvalue in $\boldsymbol{A}_e$. Then $\forall i \in \{1, \ldots, n\}. |\lambda_{oi} - \hat{\lambda}_{oi}| < \delta_o$, where λ_{oi} are the true eigenvalues of the observer $\boldsymbol{A}_o$ and $\hat{\lambda}_{oi}$ the corresponding eigenvalues of the closed loop. We note that in the case of stability we only care that their norms are smaller than one, which is

equivalent to upper bounding the estimation of the largest eigenvalue. Hence, given a desired upper value $\overline{\lambda}$ (see Section 5.3.3), we apply Jury's criterion on the polynomial

$$P'_L(z) = \sum_{j=0}^{p} c_j \overline{\lambda'}^{p-j} z^j, \text{ where } \overline{\lambda'} = (1 + \delta_o)\overline{\lambda} \text{ and } c_j = l_j - a_j. \tag{5.6}$$

In the case of $\boldsymbol{A}_c$, the procedure is the same except that we need to look at the columns shared with $\boldsymbol{A}_c$. Thus we obtain our bound $\delta_c = \max(\|\boldsymbol{A}_{e*,i} + \boldsymbol{B}_{e*,i}\|_1)$ (i.e., the largest column norm in $\begin{bmatrix} \boldsymbol{A}_e \\ \boldsymbol{B}_e \end{bmatrix}$) and apply Jury's criterion over

$$P'_K(z) = \sum_{j=0}^{p} c_j \overline{\lambda'}^{p-j} z^j, \text{ where } \overline{\lambda'} = (1 + \delta_c)\overline{\lambda} \text{ and } c_j = k_j - a_j. \tag{5.7}$$

5.3.2 Safety of a digitally observed system

A safety specification leads to a requirement on the states of the model. Consequently, the controller must ensure that the state never violates the requirement. In this work, the safety property is expressed as:

$$\begin{aligned} \phi^n_{safety} &:= \forall k \in \{0, \ldots, n\}, \boldsymbol{G}\boldsymbol{x}_k \leq \boldsymbol{h} \text{ and } \underline{y} \leq y_k \leq \overline{y}, \\ \phi_{safety} &:= \phi^{\infty}_{safety}, \end{aligned} \tag{5.8}$$

where $\boldsymbol{G}$ and $\boldsymbol{h}$ define a set of inequalities that describe a safe set, within the state space $\mathbb{R}^p$. FWL effects are not explicitly captured here as they are implicit from Equation (5.5).

Furthermore, it is practically relevant to consider the constraints ϕ_{input} on the input signal u_k and ϕ_{init} on the initial states x_0, which we assume have given bounds, namely $\phi_{input} := \underline{u} \leq u_k \leq \overline{u}$, which come from physical restrictions in the energy of the system; and $\phi_{init} \equiv \boldsymbol{G}'\boldsymbol{x}_0 \leq \boldsymbol{h}'$, where $\boldsymbol{G}'$ and $\boldsymbol{h}'$ limits the initial conditions of the model.

5.3.3 Transient response

In many cases the response time of the model, i.e., how fast it converges to the set point, is a high-priority requirement. This is given as a "settling time".

Theorem 7. *Given a set of eigenvalues for a discrete time model of a p^{th} order stable system $\{\lambda_i = e^{\omega_i} \mid i \in \{1, \ldots, p\}, |\lambda_i| \leq 1\}$, the settling time of the system is upper-bounded by*

$$t_s < -\log_{\overline{\lambda}}(P_s), \text{ where } \overline{\lambda} = \max(|\lambda_i|) \text{ and } \overline{\lambda} < 1, \ \forall i \in \{1, \ldots, p\}, \tag{5.9}$$

where $0 \leq P_s \leq 1$ is the desired portion of the target range (the difference between the initial state and the final state measured at the output [1]) within which the output must remain after t_s.

Proof. In a SISO system, there are two values that typically define the response-time of the model (using a step response).

1. The *rise time* is defined as the time it takes the output to reach a percentage (e.g., 90%) of the target range given a step change in the input.

2. The *settling time* is defined as the time it takes the output to stabilise within a margin of the target value (e.g., 10% of the target range).

Given a closed loop single output LTI model of order p with dynamics

$$\begin{cases} \boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x}_{k-1} \\ \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k \end{cases} \Rightarrow \begin{cases} \boldsymbol{x}'_k = \boldsymbol{J}\boldsymbol{x}'_{k-1} \\ \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{S}\boldsymbol{x}'_k \end{cases}, \text{ where } \begin{cases} \boldsymbol{A} = \boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1} \\ \boldsymbol{x}'_k = \boldsymbol{S}^{-1}\boldsymbol{x}_k \end{cases}, \tag{5.10}$$

the output decay is a linear combination of the decay of eigenvalues or eigenvalue pairs. Let $\boldsymbol{C}' \in \mathbb{R}^{1\times p} = \boldsymbol{C}\boldsymbol{S}$. Since $\boldsymbol{C}'$ is constant, the output y_k is a linear combination of the

[1] Since our final state is always 0, this is equivalent to a percentage of the initial state.

states $\boldsymbol{x}'_k$. We accelerate Equation (5.10) to obtain

$$y_k = \boldsymbol{C'}\boldsymbol{J}^k\boldsymbol{x}_0 = \sum_{i=0}^{p}\left(c_i\lambda_i^k\boldsymbol{x}_{0i} + \sum_{j=1}^{gm(i)} c_{i+j}\binom{k}{j}\lambda_i^{k-j}\boldsymbol{x}_{0j}\right),$$
$$|y_k| \leq \sum_{i=0}^{p}\left(c_i\boldsymbol{x}_{0i} + \sum_{j=1}^{gm(i)} c_{i+j}\binom{k}{j}\lambda_i^{-j}\boldsymbol{x}_{0j}\right)|\lambda_i^k|,$$

where the second part of the equation corresponds to the upper diagonal terms of the Jordan blocks. We also have $|\lambda_i^k| \leq |\overline{\lambda}^k|$, where $|\overline{\lambda}^k|$ is the norm of the largest eigenvalue in the system. Hence

$$|y_k| \leq |\overline{\lambda}^k| \sum_{i=0}^{p}\left|c_i + \sum_{j=1}^{gm(i)} c_{i+j}\binom{k}{j}\lambda_i^{-j}\right| .$$

In the case of closed loop forms without geometric multiplicity (which is the most common case), the right-hand terms disappear and we obtain

$$|y_k| \leq |\overline{\lambda}^k| \sum_{i=0}^{p}|c_i| .$$

Hence, the overall decay of the model $P_k = \frac{|y_k|}{|y_0|}$ is always smaller than $|\overline{\lambda}^k|$ (the constants cancel out in the division). Reversing, we have $log_{|\overline{\lambda}|}(P_k) \leq k_s$, which replacing for the sample time $t_s = -k$ results in $t_s \leq -log_{|\overline{\lambda}|}(P_s)$ as in Equation (5.9).

In the case where we allow for geometric multiplicities, we need to compensate for the effects of $\binom{k}{j}$ [2]. We recall here that our interest is to reach within 10% of the final value, and that for a given $k > \underline{k}$ the term λ^k will converge faster than $\binom{k}{j}$ diverges. The requirements for $|\overline{\omega}|$ are calculated as follows:

- Find $\underline{k} = \frac{t_s}{T_s}$ where t_s is the settling time and T_s the sampling time.

[2]Tighter bounds can be obtained by evaluating $\frac{\sum_{j=1}^{gm(i)} c_{i+j}\binom{k}{j}\lambda_i^{-j}\boldsymbol{x}_{0j}}{\sum_{j=1}^{gm(i)} c_{i+j}\boldsymbol{x}_{0j}}$

- Calculate $|\overline{\lambda}| = \log_{t_s}\left(P_s\left(\frac{k}{gm-1}\right)^{-1}\right)$, where gm is the maximum desired geometric multiplicity.
- Ensure that $|\overline{\lambda}| \leq \frac{k}{k+1}$ and that the synthesised closed loop does not have a larger gm than specified.

□

We now use this limit to express a constraint in terms of the coefficients of the characteristic polynomial $P'_K(z)P'_L(z)$. Let this polynomial be:

$$P(z) = \prod_{i=0}^{p}(z - \lambda_i) = \sum_{j=0}^{p} c_j z^j. \tag{5.11}$$

Let us now define a transformation $\lambda'_i = \frac{\lambda_i}{\overline{\lambda}}$. We want to ensure $\lambda'_i \leq 1$, which can be done by applying Jury's criterion to $P'(z) = \prod_{i=0}^{p}(z - \lambda'_i) = \sum_{j=0}^{p} c'_j z^j$ where c'_j are unknown. By replacing λ_i, we obtain

$$P'(z) = \prod_{i=0}^{p}(z - \lambda'_i) = \prod_{i=0}^{p}\left(z - \frac{\lambda_i}{\overline{\lambda}}\right) = \sum_{j=0}^{p} \frac{c_j}{\overline{\lambda}^{p-j}} z^j, \tag{5.12}$$

where all values are known. Thus, we can ensure that the settling time is smaller than a certain magnitude by ensuring that the eigenvalues are smaller too (which we can do with Jury's criterion).

We will use $\phi_{stability}$ to refer to the combined stability and transient response specifications.

5.4 Synthesising digital controllers using CEGIS

In this section, we describe our technique for synthesizing safe digital feedback controllers. Initially, we discuss two instantiations of CEGIS for stateless controllers: the first, naïve

one, relies on an unfolding of the dynamics up to a completeness threshold [3], while the second one is abstraction-based and leverages abstraction refinement and acceleration to improve scalability while retaining soundness. We then take a step further by introducing an optimiser on the abstraction-based method. To demonstrate the effectiveness of this approach, we increase the complexity of the problem by synthetizing stateful controllers. The controllers described in the first two section asume full observability of the state, which is in most cases infeasible. Controllers that predict the internal state from the outputs are much harder to synthesise, not only because they are of higher order, but also because the estimation errors require larger safe regions.

5.4.1 Naïve approach

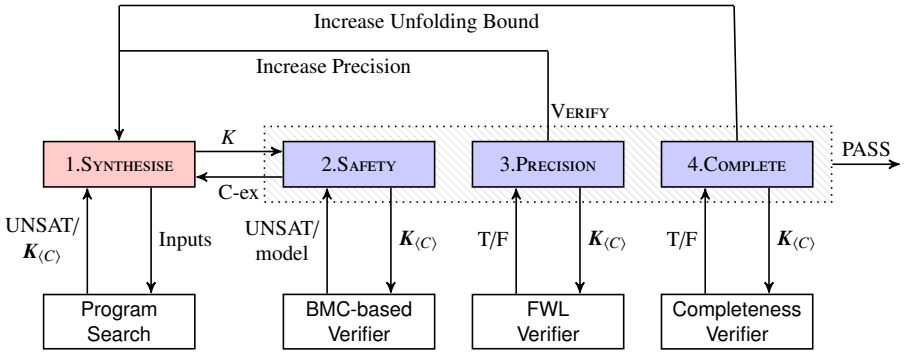

Figure 5.2: CEGIS with multi-staged verification

An overview of the algorithm for controller synthesis is given in Fig. 5.2. One important observation is that we verify and synthesise a controller over k time steps. We then compute a completeness threshold $\bar{k}$ [119] for this controller, and verify correctness for $\bar{k}$ time steps. Essentially, $\bar{k}$ is the number of iterations required to sufficiently unwind the closed-loop state-space system, which ensures that the boundaries are not violated for any other $k>\bar{k}$.

Next, we describe the different phases in Fig. 5.2 (blocks 1 to 4) in detail.

[3]The naïve approach is presented here as a basis for comparison of the developed techniques, and is as such not part of the results of this thesis.

1. The inductive synthesis phase (SYNTHESISE) uses BMC to compute a candidate solution $\boldsymbol{K}_{\langle C \rangle}$ that satisfies both the stability criteria and the safety specification. To synthesise a controller that satisfies the stability criteria, we require that a computed polynomial satisfies Jury's criterion [74] (See Section 2.6.3).

   Regarding the second requirement, we synthesise a safe controller by unfolding the transition system k steps and by picking a controller $\boldsymbol{K}_{\langle C \rangle}$ and a single initial state, such that the states at each step do not violate the safety criteria. That is, we ask the bounded model checker if there exists a $\boldsymbol{K}_{\langle C \rangle}$ that is safe for at least one $\boldsymbol{x}_0$ in our set of all possible initial states. This is sound if the current k is greater than the completeness threshold. We also assume some (fixed or floating point) precision $\mathbb{R}_{\langle P \rangle}$ for the plant and a sampling rate. The checks that these assumptions hold are performed by subsequent VERIFY stages.

Algorithm 2 Safety check

```
1:  function safetyCheck()
2:      assert(u ≤ u ≤ ū)
3:      set x to be a vertex of the initial state, e.g., x = [x_0 ··· x_0]
4:      for (c = 0; c < 2^Num_States; c++) do
5:          for (i = 0; i < k; i++) do
6:              u = F_⟨P⟩((F_⟨C⟩(K) * F_⟨C⟩(x))
7:              x = A * x + B * u
8:              assert(∀ j ∈ [1, ..., p], x ≤ x_j ≤ x̄ )
9:          end for
10:         set x_0 to be a new vertex state
11:     end for
12: end function
```

2. The first VERIFY stage, SAFETY, checks that the candidate solution $\boldsymbol{K}_{\langle C \rangle}$, which we synthesised to be safe for at least one initial state, is safe for *all* possible initial states, i.e., does not reach an unsafe state within k steps where we assume k to be under the completeness threshold. After unfolding the transition system corresponding to the previously synthesised controller k steps, we check that the safety specification holds for any initial state. This is shown in Alg. 2.

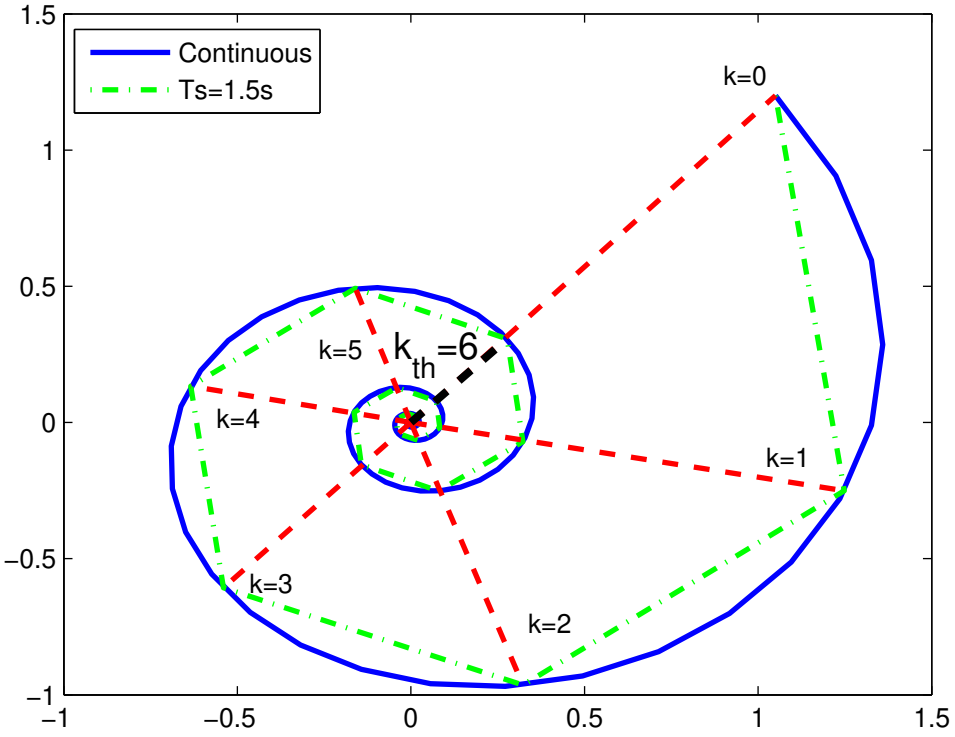

Figure 5.3: Completeness threshold for multi-staged verification. T_s is the time step for the time discretisation of the control matrices.

3. The second VERIFY stage, PRECISION, restores soundness with respect to the plant's precision by using interval arithmetics [132] to validate the operations performed by the previous stage.

4. The third VERIFY stage, COMPLETE, checks that the current k is large enough to ensure safety for any $k'>k$. Here, we compute the completeness threshold $\overline{k}$ for the current candidate controller $\boldsymbol{K}_{\langle C \rangle}$ and check that $k \geq \overline{k}$. This is done according to the argument given above and illustrated in Fig. 5.3.

Checking that the safety specification holds for any initial state can be computationally expensive if the bounds on the allowed initial states are large.

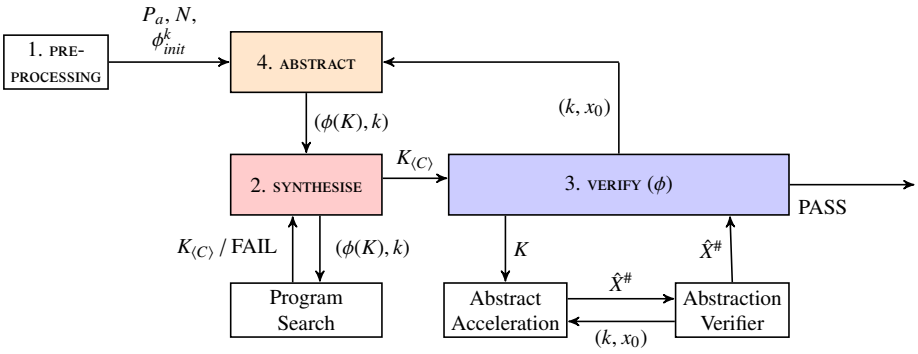

Figure 5.4: CEGIS-AR: Abstraction-based CEGIS.

5.4.2 Abstraction-based CEGIS

Essentially, the naïve approach relies on the symbolic simulation over a bounded time horizon of individual initial states and inputs that form part of an uncountable space and tries to generalise it for an infinite space over an infinite time horizon.

Conversely, in this section, we find a controller for a continuous initial set of states and set of inputs, over an abstraction of the continuous dynamics [42] that conforms to witness proofs at specific times. Moreover, this approach uses abstraction refinement enabling us to always start with a very simple description regardless of the complexity of the overall dynamics, and only expand to more complex models when a solution cannot be found.

The CEGIS loop for this approach, which we refer to as CEGIS-AR, is illustrated in Fig. 5.4.

1. We start by doing some preprocessing:

   (a) Compute the characteristic polynomial of the matrix $(\boldsymbol{A}_d - \boldsymbol{B}_d \boldsymbol{K}_{\langle C \rangle})$ as $P_a(z) = z^n + \sum_{i=1}^{n} (a_i - k_i) z^{n-i}$. As with the naïve approach, this polynomial must satisfy Jury's criterion.

   (b) Calculate the noise set N from the quantiser resolutions and estimated round-off errors:

$$N = \left\{ \nu_1 + \nu_2 + \nu_3 \mid \nu_1 \in \left[-\frac{q_1}{2} \;\; \frac{q_1}{2} \right], \;\; \nu_2 \in \left[-\frac{q_2}{2} \;\; \frac{q_2}{2} \right], \nu_3 \in \left[-q_3 \;\; q_3 \right] \right\}$$

where q_1 is the error introduced by the truncation in the ADC (Q1 in Figure 5.1), q_2 is the error introduced by the DAC (Q2 in Figure 5.1) and q_3 is the maximum truncation and rounding error in $u_k = -\boldsymbol{K}_{\langle C \rangle} \cdot \mathcal{F}_{\langle C \rangle}(\boldsymbol{x}_k)$ as discussed in Section 2.3.1.

(c) Calculate a set of initial bounds on $\boldsymbol{K}_{\langle C \rangle}$, ϕ^K_{init}, based on the input constraints

$$\phi^K_{init} := (\phi_{init} \wedge \phi_{input} \wedge u_k = -\boldsymbol{K}_{\langle C \rangle} x_k)$$

Note that these bounds will be used by the SYNTHESISE phase to reduce the size of the solution space.

2. In the SYNTHESISE phase, we synthesise a candidate controller $\boldsymbol{K}_{\langle C \rangle} \in \mathbb{R}^n_{\langle C \rangle}$ that satisfies $\phi_{stability} \wedge \phi_{safety} \wedge \phi^K_{init}$ by invoking a SAT solver. If there is no candidate solution we return UNSAT and exit the loop.

3. Once we have a candidate solution, we perform a safety verification of the progression of the system from ϕ_{init} over time, $\boldsymbol{x}_k \models \phi_{safety}$. In order to compute the progression of point x_0 at iteration k, we accelerate the dynamics of the closed-loop system and obtain:

$$x = (\boldsymbol{A}_d - \boldsymbol{B}_d \boldsymbol{K}_{\langle C \rangle})^k \boldsymbol{x}_0 + \sum_{i=0}^{k-1} (\boldsymbol{A}_d - \boldsymbol{B}_d \boldsymbol{K}_{\langle C \rangle})^i \boldsymbol{B}_n (\nu_1 + \nu_2 + \nu_3) \tag{5.13}$$

$$B_n = [1 \cdots 1]^\intercal \tag{5.14}$$

As this still requires us to verify the system for every k up to infinity, we use abstract acceleration again to obtain the reach-tube, i.e., the set of all reachable states at all times given an initial set ϕ_{init}:

$$\hat{X}^\# = \mathcal{A} X_0 + \mathcal{B}_n N, \quad X_0 = \{\boldsymbol{x} \mid \boldsymbol{x} \models \phi_{init}\}, \tag{5.15}$$

where $\mathcal{A} = \bigcup_{k=1}^{\infty}(\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K}_{\langle C\rangle})^k, \mathcal{B}_n = \bigcup_{k=1}^{\infty}\sum_{i=0}^{k}(\boldsymbol{A}_d - \boldsymbol{B}_d\boldsymbol{K}_{\langle C\rangle})^i\boldsymbol{B}_n$ are abstract matrices for the closed-loop system [42], whereas the set N is non-deterministically chosen.

We next evaluate $\hat{X}^{\#} \models \phi_{safety}$. If the verification holds we have a solution, and exit the loop. Otherwise, we find a counterexample iteration k and corresponding initial point x_0 for which the property does not hold, which we use to locally refine the abstraction. When the abstraction cannot be further refined, we provide them to the ABSTRACT phase.

4. If we reach the ABSTRACT phase, it means that the candidate solution is not valid, in which case we must refine the abstraction used by the synthesiser.

   (a) Find the constraints that invalidate the property as a set of counterexamples for the eigenvalues, which we define as ϕ_Λ. This is a constraint in the spectrum i.e., transfer function) of the closed loop dynamics.

   (b) We use ϕ_Λ to further constrain the characteristic polynomial $z^n + \sum_{i=1}^{n}(a_i - k_i)z^{n-i} = \prod_{i=1}^{n}(z - \lambda_i)$ where $|\lambda_i| < 1 \wedge \lambda_i \models \phi_\Lambda$. These constraints correspond to specific iterations for which the system may be unsafe.

   (c) Pass the refined abstraction $\phi(K)$ with the new constraints and the list of iterations k to the SYNTHESISE phase.

5.4.3 Using optimisation refinement for CEGIS of control systems

We now enhance the CEGIS architecture with abstraction-refinement by adding an optimisation step. We call the resulting new technique CEGIS-OR (CEGIS with Optimisation and Refinement). By using abstraction refinement, CEGIS-OR addresses the scalability issues inherent in program synthesis and enables us to handle complex models, which we coarsely abstract and only refine on demand – refinements are triggered by counterexamples returned by the verification phase. Since the use of coarse abstractions may result in low-quality candidate solutions (i.e. candidates that fail the specification on a very large number of inputs), after each synthesis phase we use optimisation techniques

to improve the produced candidates. In particular, our optimisation procedure aims at reducing the size of the reach space of the closed loop model, while maintaining other dynamical properties (safety, stability, and response time).

The main advantage of the new CEGIS-OR is twofold:

1. We start with coarse abstractions that get refined into more complex ones as the procedure progresses. Thus, solutions that do not require a high level of precision are obtained faster. Note that the optimisation step plays an important role here given that, in its absence, the use of coarse abstractions often results in low-quality candidate solutions (i.e., candidates that fail the specification on a very large number of inputs) and, consequently, there are very few solutions obtained at such low levels of precision.

2. Although SYNTHESISE makes use of a potentially coarse abstraction, by optimizing the candidate solution in each iteration of the CEGIS loop we obtain better candidates that are more likely to meet the specification or produce high quality counterexamples in the VERIFY phase. This has the potential to reduce the overall number of iterations of the CEGIS loop required to obtain a complete solution.

Throughout the CEGIS-OR loop we make use of a specification ϕ that deals with the abstraction of the model dynamics. Note that since the model abstraction gets refined at each iteration, ϕ will also be adjusted accordingly. Initially, ϕ is simply given as the defined $\phi := \phi_{stability} \wedge \phi_{input} \wedge \phi_{init}$.

Note that there is a major difference between the way ϕ is being used in different phases in Figure 5.5: while the SYNTHESISE phase checks ϕ by using the plant precision $\mathbb{R}_{\langle P \rangle}$ for the controller, observer and plant, the MODEL and VERIFY phases use infinite precision ($\mathbb{R}$) to represent the plant and $\mathbb{R}_{\langle C \rangle}$ for the controller and the observer. The reason for this is that, as we want the synthesis to be fast, we allow it to generate unsound results and we restore soundness during the verification and modelling stages. Since we use different precisions at different stages, we will often use $\boldsymbol{K}$ and $\boldsymbol{L}$ to indicate that the domain they belong to is unspecified (these are always truncated to the corresponding domain at each stage).

Next, we describe the salient blocks in the CEGIS-OR loop of Figure 5.5.

ABSTRACT computes an initial version of ϕ and decides the sample time T_s, the plant precision $\mathbb{R}_{\langle P \rangle}$ and the controller precision $\mathbb{R}_{\langle C \rangle}$.

- Select a sample time (T_s) that minimises the magnitudes of the used matrices. This ensures that a small word length for the plant precision is required for the synthesis process.
- Select initial precisions $\mathbb{R}_{\langle P \rangle}$ and $\mathbb{R}_{\langle C \rangle}$ for representing the controller and the plant, respectively. These will be increased as needed.
- Build the observer matrices $\boldsymbol{A}_{d\langle C \rangle}$, $\boldsymbol{B}_{d\langle C \rangle}$ and $\boldsymbol{C}_{\langle C \rangle}$.
- Transform the model into canonical form and add nondeterminism for the errors as described in Section 5.3.1.
- Calculate the noise sets η_1 and η_2 from the equations described in Section 5.3.1.
- Compute necessary characteristic polynomials.
- Create a stability specification $\phi_{stability}$ based on the characteristic polynomials, FWL error ($\boldsymbol{A}_e$,$\boldsymbol{B}_e$), and response time as described in Section 5.3.
- Calculate a set of initial bounds on $\boldsymbol{K}$, ϕ^K_{init}, based on the input constraints. ($\phi_{init} \wedge \phi_{input} \wedge u_k = -\boldsymbol{K}\boldsymbol{x}_k) \Rightarrow \phi^K_{init}$. These bounds will be used by the SYNTHESISE phase to reduce the size of the solution space.

SYNTHESISE generates a candidate controller matrix $\boldsymbol{K}_{\langle P \rangle}$ and observer matrix $\boldsymbol{L}_{\langle P \rangle}$ that satisfy ϕ by invoking a SAT solver. If there is no candidate solution it signals ABSTRACT, which may choose to increase the plant precision or return a synthesis failure. If SYNTHESISE generates a candidate solution, it is fed to the VERIFY chain in order to validate it.

VERIFY checks whether the candidate solution found by SYNTHESISE is a solution for the whole synthesis problem. It does this in three subsequent stages as explained below. If in any of the stages the verification fails, then a counterexample is fed to the REFINE ABSTRACTION AND OPTIMISE phase, including details of initial states and iterations that failed the specification.

- VERIFY SYNTHESIS verifies that the candidate controller and observer obey ϕ (remember that we now use $\mathbb{R}_{\langle C \rangle}$ for the controller representation and $\mathbb{R}$ for the plant). If the candidate is valid, we feed it to the next VERIFY stage, DISCRETE TIME.
- DISCRETE TIME verifies ϕ^{∞}_{safety} (unbounded time) using discrete time dynamics.
- CONTINUOUS TIME verifies that the transitions between sample times meet the specification ϕ^{∞}_{safety}.

REFINE ABSTRACTION AND OPTIMISE refines the model abstraction as well as optimises the current candidate solution. First, the model abstraction must be refined in order to eliminate the newly received counterexample. For this purpose, we increase the time horizon n for the safety specification so that it captures the violating iteration (for the given initial state) and adjust ϕ as $\phi := \phi \wedge \phi^{n}_{safety}$. If the counterexample is provided by the CONTINUOUS TIME stage, then we must select a smaller suitable sample time to be used for the model discretisation.

Secondly, we try to optimise the failed candidate solution. For this purpose, we seek to minimise the distance of the farthest point in the reach space to the closest point to it in the safe region $e = \max\{\min\{\boldsymbol{x} - \boldsymbol{g}, \boldsymbol{g} \in G\}, \boldsymbol{x} \in \hat{X}_{\infty}\}$.

Our optimisation is iterative, following two stages. The first stage seeks to make the dynamics as convergent as possible, since convergent dynamics are less likely to stray outside of the safe region (we say less likely because the change in the eigenvectors changes the shape of the space and may cause some trajectories to go further). For this purpose, we take the largest eigenvalue and reduce it by a factor f that still satisfies the controller constraints ϕ^{K}_{init}. The resulting observer and controller are then verified against the safety specification ϕ_{safety} and the objective e is measured. If for each iteration k, the

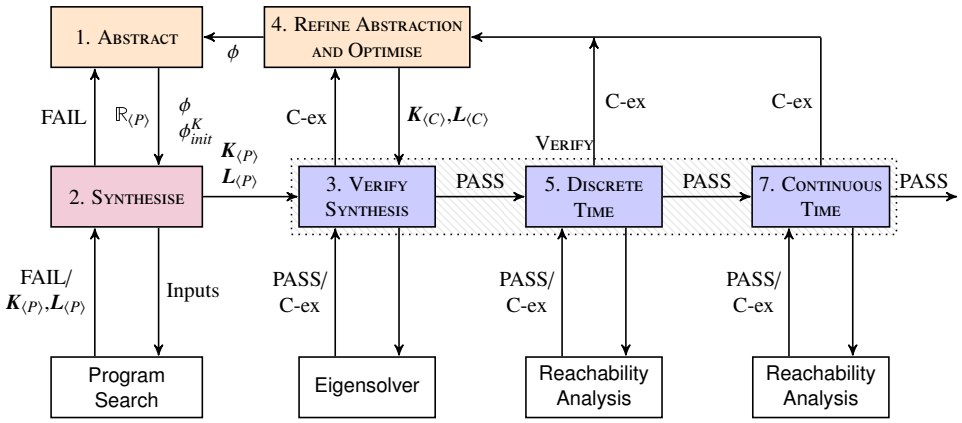

Figure 5.5: CEGIS-OR: CEGIS with Optimisation Refinement. Orange boxes (light) are modelling stages, blue boxes (dark) are verification stages, and the red box is the synthesis stage.

objective $e_k < e_{k-1}$, we continue to iterate until $e_k \leq 0$ (i.e. we found a safe controller), otherwise we move to the second stage.

In the second stage, we take the complex eigenvalue with the largest angle, which is characterised by $\max\{|\tan^{-1}\left(\frac{im(\lambda)}{re(\lambda)}\right)| \mid \lambda \in \Lambda\}$, and reduce its imaginary component only $(re(\lambda) + f\, im(\lambda)$ such that the new controller satisfies $\phi^K_{init})$. The process ends when no imaginary part can be further reduced. The resulting closed loop form converges faster than the original and has a smaller rotation, which facilitates meeting the safety specification ϕ_{safety}.

The optimiser tool uses infinite precision for the plant, and may therefore find solutions that the synthesiser cannot, due to the use of limited precision. If we cannot optimise the candidate we signal the Abstract phase to use the updated ϕ and request Synthesise to produce a new candidate. Otherwise the optimised candidate solution is given back to Verify Synthesis.

5.5 Experimental Evaluation

5.5.1 Description of the benchmarks

The benchmark suite employed to evaluate the proposed technique is composed by 15 SISO models extracted from the literature [18, 80, 87, 116, 135, 140, 155, 157, 158, 160].

In the case of stateless controllers, we have manually discretised these for different sampling times (up to 8 per benchmark), selecting the shortest time obtained for any synthesis run in each model (since these may be run in parallel).

In the case of controllers with observers we first reduced the set to seven (some of these systems are not fully observable and first order ones do not require an observer). For this second set the sampling is done automatically. In order to obtain statistically relevant results, we have taken these models and modified each possible combination of their parameters by a 10% tolerance, thus creating seven hundred benchmarks from this base.

DC Motor DC Motor benchmark describes the velocity dynamics of a basic brushed direct current electrical machine.

Helicopter Longitudinal Motion The Helicopter Longitudinal Motion benchmark describes the dynamics of transitional motion with respect to a longitudinal plane and the rotational motion around the longitudinal axis.

Automotive Cruise System An automotive cruise control system is used to keep constant the speed of an automobile tracking a desired speed reference, compensating disturbances and uncertainties related to relief and terrain.

Inverted Pendulum An inverted pendulum is a system similar to the pendulum but the point mass must be equilibrated above the pivot point that is mounted on an one degree of freedom cart that moves over a track through a DC motor.

Magnetic Suspension Magnetic suspension benchmark corresponds to the discrete model of a simple electromagnet-ball system where a steel ball is levitated in air by the action of an electromagnetic force generated by an electromagnet.

Magnetic Pointer Magnetic pointer is a pointer, that is usually employed in analogue gauges and indicators, whose angular position dynamics is controlled by a magnetic field.

Pendulum A pendulum is a system composed by a swinging point mass suspended from a frictionless pivot through a negligible mass rod.

bioreactor A linear model identified by least squares for the cell mass concentration controlled through dilution rate of a bioreactor.

Car Suspension The Car Suspension benchmark corresponds to the model of a single wheel suspension system of a car, *i.e.* the relative motion dynamics of a mass springer damp model that connects the car to one of its wheels.

Computer Tape Driver Computer Tape Drive is a computer storage device used to read and write data on magnetic tapes.

High-order Systems from Chen et. al. These benchmarks corresponds to a series of high-order system model generally used as example for order reduction techniques [160].

Continuous Stirred Tank Reactor This benchmark describes a linear model for the temperature of a exothermic reaction in a continuous flow stirred-tank reactor.

Flexible Beam A Flexible Beam is a plant usually employed in vibration studies. It is composed by a flexible metallic structure with piezo-ceramic actuators and sensors.

Benchmark	Order	Multi-staged		CEGIS-AR	
		Fixed	Float	Fixed	Float
Bioreact	3	11.2s	14.6s	.4s	.5s
Chen_ex1	3	7.3s	9.2s	t.o.	1.4s
Cruise	1	6.6s	6.5s	.3s	.3s
CruiseHSCC	1	6.6s	6.5s	.3s	.3s
Cstrtmp	1	t.o.	t.o.	.5s	t.o.
DCmotor	2	6.9s	7.8s	.4s	.5s
Flexbeam	6	t.o.	t.o.	72.1s	t.o.
Helicopter	3	13.7s	148.1s	.7s	1.1s
Invpendulum_pendang	2	9.7s	28.1s	.4s	.5s
Magpointer	2	10.1s	20.4s	.9s	1.2s
Magsuspension	2	22.1s	34.8s	.4s	.6s
Pendulum	2	8.5s	24.0s	.4s	.5s
streamdrum	3	t.o.	t.o.	.7s	1.2s
Suspension	4	17.2s	t.o.	15.3s	4.4s
Tapedriver	3	8.2s	t.o.	2.0s	1.1s

Table 5.1: Experimental results for stateless controllers indicating synthesis times of fixed ($\mathbb{R}_{\langle 11,5\rangle}$) and floating point (IEEE half precision) controllers. The order denotes the number of variables of a model.

5.5.2 Experimental results

Synthesis of stateless controllers.

In our first experiment, we compare the performance of CEGIS with Abstraction Refinement (CEGIS-AR) with a standard CEGIS model (multi-staged). We synthesise stateless controllers for discrete time plant models with no settling time or sampling delay specifications. We use a timeout of 5 minutes. The results are shown in Table 5.1.

In most cases, the CEGIS-AR approach proves to be significantly faster (typically an order of magnitude), managing to synthesise 5 more controllers than the multi-staged approach.. The notable exception is the fixed point $Chen_ex1$ benchmark, in which CEGIS-AR got stuck in a recurringly increasing refinement until it timed out due to an overly large time horizon (157 iterations). The second notable case is the fixed point *suspension* benchmark, in which CEGIS-AR required a full 64 bit precision to overcome the errors of over-approximation, thus reaching comparable times to the multi-staged approach. These results show the improvement of Abstraction Refinement over standard CEGIS.

Synthesis of stateful controllers with observers.

In order to show the improvements due to our optimisation-enhanced CEGIS, we compared our approach against the CEGIS approach with CEGIS-AR, which we have shown to

Benchmark	order	No.	CEGIS-AR				CEGIS-OR			
			Min	Median	t.o.	found	Min	Median	t.o.	found
DC Motor	2	100	1.3 s	26.9 s	22	20	1.1 s	3.8 s	2	58
Helicopter	3	100	57.2 s	t.o.	54	15	7.1 s	18.0 s	9	90
Inverted Pendulum	4	100	277 s	t.o.	82	0	20.0 s	t.o.	55	20
Magnetic Pointer	2	100	5.7 s	229.9 s	38	42	3.0 s	9.5 s	5	95
Pendulum	2	100	0.8 s	53.6 s	27	71	0.4 s	4.9 s	6	92
Suspension	4	100	15.0 s	177.2 s	30	70	14.8 s	166.9 s	26	73
Tape Driver	3	100	4.0 s	17.7 s	11	80	3.1 s	16.8 s	8	82

Table 5.2: Experimental results. Minimum, and Median times for standard CEGIS-AR (with abstraction refinement) vs CEGIS-OR on multiple benchmarks. No. indicates Number of benchmarks, t.o. indicates timeouts (in the median case, it indicates more than half of the benchmarks timed out). The order denotes the number of variables of a model.

be significantly faster than standard CEGIS. We further note that these benchmarks also include the analysis of continuous time, sampling delays, and response time specifications. We do not compare against our standard CEGIS implementation due to the fact that it cannot handle continuous time and in order to separate the effects of the optimisation from those of the abstraction refinement. For the majority of our benchmarks, we observe that our approach is significantly faster than CEGIS-AR. The results are shown in Table 5.2

Using the described technique we managed to synthesise controllers for 509 benchmarks (compared to 290 found by our baseline), with a median time of 16.8 seconds. We consider these times short enough to be of practical use to control engineers. Regarding the cases for which we fail to find a controller, we note that we use a relatively short timeout (5 minutes), so it is possible that longer timeouts would instead lead to a valid solution. However, there are cases where a solution may not be found due the approximations in the synthesiser that would prevent a true candidate from being found or over-approximations in the verifier that may reject a valid candidate. It is impossible for us to differentiate these from the case where a controller does not exist.

Overall, we observed more improvements with CEGIS-OR for models where the safety specification is tight, meaning that the CEGIS-AR loop would require many iterations. These improvements are due to the interplay between Abstraction Refinement and Optimisation in CEGIS-OR. While abstraction refinement enables us to start with coarse abstractions that are only refined on demand, the optimisation allows to locally improve candidate solutions and avoid entire iterations of the CEGIS [2] loop (which would have also required more refined abstractions). While some models might still require a very precise abstraction resulting in many refinements (this is generally the

cause of time-outs in CEGIS-OR), others allow us to find solutions with rather coarse abstractions (this is reflected in the smaller minimum times obtained with CEGIS-OR).

Chapter 6

Abstract Acceleration Using Sound Numeric Computations

6.1 Implementation

The calculation of the abstract reach tube via abstract acceleration is encompassed in Algorithm 3, which can be summarised as follows (its steps are denoted in parentheses):

1. We perform unsound eigendecomposition using an existing algebra package (lines 2-3).

2. Then we restore soundness to the results using the methods described in Section 6.1.2 (lines 4–5).

3. The inverse of the matrix of eigenvectors is calculated after soundness is restored, using interval arithmetics in order to ensure its soundness (line 6).

4. The abstract dynamics are obtained by evaluating the convex hull of all powers of eigenvalues up to the desired number of iterations as described in [42] (line 7).

5. Using Equation (4.1), transform the initial state into the eigenspace (line 8).

6. Extract the vertices of the eigen-polyhedron X'_0 using the sound over-approximation of the double description algorithm [86] from Section 6.1.4 (line 9). It is worth noting that this algorithm uses a simplex to seed it: we will first discuss the simplex algorithm, which is also needed at the next step, and then the vertex enumeration algorithm used at this stage.

7. Calculate the mapped *abstract reach tube* $\hat{X}'^{\sharp}$ that over-approximates the image of the reach tube in the eigenspace (line 10). This is achieved by evaluating a set of objective functions using the abstract dynamics as a simplex Tableau, and via a sound simplex described in Section 6.1.3. The objective functions are defined by the vertices of X'_0, denoted as V_0, and by the desired template directions, such that

$$\boldsymbol{w}_{ij} = \boldsymbol{v}_i \circ \boldsymbol{t}_j : \boldsymbol{v}_i \in V_0 \wedge \boldsymbol{t}_j \in T,$$

   where T is the set of template directions, and $\circ$ denotes a component-wise multiplication yielding a vector.

8. Inverting Equation (4.1), find the reach tube $\hat{X}^{\sharp} = \boldsymbol{S}\hat{X}'^{\sharp}$ (line 11).

Algorithm 3 Calculation of abstract reach tube using abstract acceleration

Input: $X_0\boldsymbol{A}, k$.
Output: $\hat{X}^{\sharp}$

```
 1: function findAbstractReachTube()
 2:     Ĵ = calculateEigenvalues(A)
 3:     Ŝ = calculateEigenvectors(A)
 4:     𝕁 = soundifyEigenvalues(A, Ĵ, Ŝ)
 5:     𝕊 = soundifyEigenvectors(A, 𝕁, Ŝ)
 6:     𝕊^-1 = calculateInverse(𝕊)
 7:     𝒥 = getAbstractDynamics(𝕁)
 8:     [X'_0] = transformInitialSpace(X_0, 𝕊^-1)
 9:     [V_0] = getVertices([X'_0])
10:     X̂'♯ = getReachTube(𝒥, [V_0])
11:     X̂♯ = transformReachTube(X̂'♯, 𝕊)
12: end function
```

In this chapter we will refer to this algorithm to explain the means of implementing a sound floating point implementation.

6.1.1 An interval partial order for vector spaces

Interval arithmetics are widely researched and used in the literature. The main issue in its implementation here is with respect to ordering, since some of the operations used require ordered sets. While real numbers have a well defined order, there are several different options regarding the order of real intervals. This creates a problem for programs dealing with branching since an incorrect assumption on the order will cause undetermined behaviour. The reader is referred to literature [39, 64, 76, 130] for possible orderings of real intervals. In this paper, we have selected the following paradigm:
Let $[x] = [\underline{x}, \overline{x}]$ be an interval of real numbers such that

$$\begin{cases} [x] < 0 & \overline{x} < 0 \\ [x] > 0 & \underline{x} > 0 \\ [x] = 0 & -e \leq \underline{x} \leq 0 \wedge 0 \leq \overline{x} \leq e \\ [x] \text{ is deemed imprecise} & \underline{x} < -e \wedge \overline{x} \geq 0 \vee \underline{x} \leq 0 \wedge \overline{x} > e, \end{cases} \tag{6.1}$$

where e is a user-defined error bound.

The first two definitions correspond to precedence definitions in the IEEE standard, but the third one is an enabling comparison (*i.e.* it corresponds to "may be equal to") which is not present in the standard. Throughout this work this latter definition is useful because the operations that relate to this condition have no negative effect if applied when the condition is false (apart from the increased processing time of the operation), nor do they compromise soundness.

Imprecise numbers break the ordering: for example, they could be originally non-zero elements whose error touches zero – while this situation does not break the soundness of the algorithms described in this paper, it affects their precision and completeness, as described in Section 6.1.3). As such, it is established that the appearance of an imprecise number forces a change in the error bound e or an increase in the precision, so that the accumulated errors do not surpass this bound (e). From the above paradigm

we can easily derive that

$$\begin{cases} [x] = [y] & \iff [x] - [y] = 0 \\ [x] < [y] & \iff [x] - [y] < 0 \\ [x] > [y] & \iff [x] - [y] > 0, \end{cases} \tag{6.2}$$

which results in our ordering.

We will extend this definition to the dot product of two vectors, in order to establish equality in higher-dimensional spaces (in particular, this will allow us to set an ordering of value pairs). Let $\mathbb{v} = [[v_1] \cdots [v_n]]^\intercal$ and $\mathbb{u} = [[u_1] \cdots [u_n]]^\intercal$ be interval column vectors, with $[d] = \mathbb{v} \cdot \mathbb{u}$, then we say that

$$\begin{cases} \mathbb{v} \cdot \mathbb{u} < 0 & \overline{d} < 0 \\ \mathbb{v} \cdot \mathbb{u} > 0 & \underline{d} > 0 \\ \mathbb{v} \cdot \mathbb{u} = 0 & -e \leq \underline{d} \leq 0 \wedge 0 \leq \overline{d} \leq e \\ \mathbb{v} \cdot \mathbb{u} \text{ is deemed imprecise} & \text{otherwise.} \end{cases} \tag{6.3}$$

6.1.2 Eigendecomposition

The first stage required for abstract acceleration is the eigen-decomposition of the dynamics (steps 2-3 in Algorithm 3). We seek to find the Jordan form of a matrix, characterising its eigenvalues alongside their corresponding eigenvectors, with known error bounds for both. For this purpose we use the package *eigen* [95], which contains an efficient fast numerical eigensolver using Schur decomposition. The main advantage of eigen over other packages is that it is a template library which can be used with any numerical data type. Since we are interested in using multiple precision floating point integers, this is an important feature of the desired package. Unfortunately, the eigendecomposition cannot be performed using interval arithmetics. This is because numerical solvers for this problem exploit the convergence of successive results towards a precise value. While the process is known to converge, the latter iterations will typically oscillate around the final values, which causes the interval containing the final eigenvalue to expand, resulting in a width that becomes unbounded, rather than in a small interval around the expected result. We therefore use standard arithmetics to obtain a numerical approximation of

the true eigenspace, and then find error bounds using interval arithmetic to generate the intervals containing the true values in the eigenspace. Namely, we call a standard unsound eigen-decomposition algorithm, and later make it sound by creating intervals around the results using soundly-calculated error bounds.

We remark that these error bounds are found in the LAPACK [11] package, used by programs such as Matlab, and could therefore be correctly obtained by using this tool. However, four issues drive us away from LAPACK. The first is that the library is written in FORTRAN and uses double-precision floating-point arithmetic: this restricts our ability to use higher precision to obtain smaller errors. The second is that some of the procedures used in LAPACK to calculate the error can be time consuming, which can have a large impact on the overall processing time. The third one is that LAPACK does not allow the use of intervals, hence the operations that calculate the error bound have rounding errors themselves and are thus unsound. The final problem is that LAPACK does not provide Jordan forms with geometric multiplicities, nor can it always ensure the error bounds for algebraic multiplicities greater than one.

The calculation of the error bounds for soundness is performed in two stages, first for the eigenvalues and then for the eigenvectors (the latter requires the sound eigenvalue intervals in its calculation).

We first define an interval matrix, which will be used during both stages. Let

$$\mathbb{M} \in \mathbb{R}^{p\times q} = \begin{bmatrix} [m_{11}] & \cdots & \left[m_{1q}\right] \\ \vdots & \ddots & \vdots \\ \left[m_{p1}\right] & \cdots & \left[m_{pq}\right] \end{bmatrix}, \tag{6.4}$$

where $\left[m_{ij}\right] = \left[\underline{m_{ij}}, \overline{m_{ij}}\right], i \in [1, \ldots, p]$ and $j \in [1, \ldots, q]$ be an interval matrix. Interval arithmetics between interval matrices derive directly from the operations between their elements. We may trivially construct an interval equivalent of any non-interval matrix using the following definition:

$$\mathbb{M} = \boldsymbol{M} \text{ iff } \forall [m_{ij}] \in \mathbb{M},\ \underline{m_{ij}} = \overline{m_{ij}} = m_{ij} \in \boldsymbol{M} \text{ with } i \in [1, \ldots, p] \text{ and } j \in [1, \ldots, q]. \tag{6.5}$$

Error bounds on the eigenvalues

(step (4) in Algorithm 3).

Theorem 8. *Given an eigenvalue λ_i with algebraic multiplicity m_i obtained using Jordan decomposition, the error of the numerically calculated eigenvalue $\hat{\lambda}_i$ is upper bounded by the formula:*

$$e_{\lambda_i} \leq e_{m_i} = \begin{cases} E & m_i = 1 \\ \frac{E^{\frac{1}{m_i}}}{1-E^{\frac{1}{m_i}}} & m_i > 1 \end{cases}, \text{ where } E = \max\left(k\left(\hat{\mathbb{S}}\right)\|\hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}}^{-1} - \mathbb{A}\|_2\right), \tag{6.6}$$

where m_i is the geometric multiplicity of λ_i, $\hat{\mathbb{S}} = \hat{\boldsymbol{S}}$ are the calculated eigenvectors of $\boldsymbol{A}$, $\hat{\mathbb{J}} = \hat{\boldsymbol{J}}$ its calculated Jordan form and $k(\mathbb{M})$ is the condition number of a given matrix $\mathbb{M}$ that is defined as $[\sigma_{max}](\mathbb{M})/[\sigma_{min}](\mathbb{M})$, where $[\sigma_i]$ are the singular values of $\mathbb{M}$.[1] *The obtained matrix $\mathbb{J} = [\hat{\boldsymbol{J}} - \sup(e_{m_i})\boldsymbol{I}, \hat{\boldsymbol{J}} + \sup(e_{m_i})\boldsymbol{I}] \supseteq \boldsymbol{J}, i \in [1, \ldots, n]$ is a sound over-approximation of the diagonal matrix with the eigenvalues of $\boldsymbol{A}$. All matrices are in $\mathbb{R}^{n\times n}$.*

Proof. Let us first assume that matrix $\boldsymbol{A}$ is diagonalizable (an hypothesis relaxed below). Let us also define the numerically calculated approximation $\hat{\boldsymbol{A}} \simeq \boldsymbol{A}$ with Jordan form $\hat{\boldsymbol{J}}$ and eigenvectors $\hat{\boldsymbol{S}}$. Then the error bound for each of the eigenvalues is [147]:

$$e_{\lambda_i} = |\lambda_i - \hat{\lambda}_i| < k(\boldsymbol{S})\|\boldsymbol{A} - \hat{\boldsymbol{A}}\|_2 = k(\boldsymbol{S})\|\boldsymbol{A} - \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}\|_2. \tag{6.7}$$

Note that symmetrically

$$e_{\lambda_i} = e_{\hat{\lambda}_i} = |\hat{\lambda}_i - \lambda_i| < k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{A}} - \boldsymbol{A}\|_2 = k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1} - \boldsymbol{A}\|_2, \tag{6.8}$$

as long as $\hat{\boldsymbol{A}} = \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}$ has no rounding errors. Therefore, to ensure soundness, we must translate the error calculation into an interval arithmetics problem.

[1] Note that this is equivalent to $k(\mathbb{M}) = \|\mathbb{M}\|_2\|\mathbb{M}^{-1}\|_2$.

Using (6.5), Equation (6.8) then becomes

$$e_{\lambda_i} < \sup\left(k\left(\hat{\mathbb{S}}\right)\|\hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}}^{-1} - \mathbb{A}\|_2\right).$$

For simplicity, we will hereon use $\hat{\mathbb{A}} = \hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}}^{-1}$.

We could calculate tighter bounds for each eigenvalue using the condition number of individual eigenvectors (which is defined for non-square matrices as $k(\boldsymbol{M}) = \|\boldsymbol{M}\|_2\|\boldsymbol{M}^+\|_2$, where $\boldsymbol{M}^+$ is the pseudo-inverse of $\boldsymbol{M}$), but we choose this faster approach expecting the increased error to be negligible with respect to the dynamics.

Extending our analysis by relaxing the assumption made above, when there exists a Jordan block in $\boldsymbol{A}$ with geometric multiplicity m_i, then the error can be derived by leveraging [147] as follows:

$$\begin{aligned}
&\tfrac{|\lambda_i-\hat{\lambda}_i|^{m_i}}{(1+|\lambda_i-\hat{\lambda}_i|)^{m_i-1}} = (1+|\lambda_i-\hat{\lambda}_i|)\left(\tfrac{|\lambda_i-\hat{\lambda}_i|}{(1+|\lambda_i-\hat{\lambda}_i|)}\right)^{m_i} < k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1} - \boldsymbol{A}\|_2 \\
&\Rightarrow \tfrac{|\lambda_i-\hat{\lambda}_i|}{(1+|\lambda_i-\hat{\lambda}_i|)} < \left(k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1} - \boldsymbol{A}\|_2\right)^{\frac{1}{m_i}} < \sup\left(k(\hat{\mathbb{S}})\|\hat{\mathbb{A}} - \mathbb{A}\|_2\right)^{\frac{1}{m_i}} \\
&\Rightarrow e_{\lambda_i} < \sup\left(\frac{\left(k(\hat{\mathbb{S}})\|\hat{\mathbb{A}}-\mathbb{A}\|_2\right)^{\frac{1}{m_i}}}{1-\left(k(\hat{\mathbb{S}})\|\hat{\mathbb{A}}-\mathbb{A}\|_2\right)^{\frac{1}{m_i}}}\right).
\end{aligned}$$

However, this bound requires that the correct Jordan shape be selected (i.e., the one that corresponds to the original dynamics without numerical errors), which means we need to use the formula using the largest possible Jordan block for each set of similar λ (i.e., eigenvalues which intersect given their error intervals). In fact, this is not enough to ensure the bound since different shapes will result in different condition numbers (since they will have different generalised eigenvectors), so we are forced to calculate the maximum bound for all options. We will see later how to overcome this difficulty in a more efficient way. □

Now that we can obtain sound eigenvalues, we will proceed to restore soundness to the eigenvectors.

Error bounds on the eigenvectors

(step 5 in Algorithm 3).

Theorem 9. *The interval eigenvector*

$$\mathbb{v}_i = \left[\hat{\mathbb{v}}_i \cos([\theta]_i),\ \frac{\hat{\mathbb{v}}_i}{\cos([\theta]_i)}\right] \supseteq \boldsymbol{v}_i,\ \textit{where} \tag{6.9}$$
$$[\theta]_i \leq [\hat{\theta}]_i = \left(\frac{n}{n-1}\right)^{\frac{n-1}{2}} \frac{\|\mathbb{A} - \hat{\mathbb{A}}]\|_2\,(n\|\mathbb{U} - [\lambda_i]\boldsymbol{I}\|_2)^{n-1}}{n^{\frac{n}{2}} \prod_{i\neq j}\left([\lambda_j] - [\lambda_i]\right)^{m_i}} \textit{ and } [\hat{\theta}]_i < \frac{\pi}{2},$$

is an over-approximation of the true eigenvector $\boldsymbol{v}_i$. Here, n is the dimension of $\boldsymbol{A}$, m_i is the size of the i^{th} Jordan block of $\boldsymbol{A}$, $\hat{\mathbb{v}}_i = \hat{\boldsymbol{v}}_i$ the numerically calculated i^{th} eigenvector of $\boldsymbol{A}$, $[\lambda_i]$ the error-bound interval for the i^{th} eigenvalue of $\boldsymbol{A}$ (inherited from above), and $\mathbb{U} = \mathbb{Q}^{-1}\mathbb{A}\mathbb{Q}$ where $\mathbb{Q} = \boldsymbol{Q}$ the Schur decomposition of $\boldsymbol{A}$.

Given sufficient precision in the numerical calculations, we have that $[\hat{\theta}]_i < \frac{\pi}{2}$. This inequality can always be obtained by increasing precision.

Proof. The error angle between the numerically calculated i^{th} eigenvector and the true i^{th} eigenvector of $\boldsymbol{A}$ is

$$\theta_i = \cos^{-1}(\boldsymbol{v}_i \cdot \hat{\boldsymbol{v}}_i) < \frac{\|\boldsymbol{A} - \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}\|_2}{sep_i},\ \text{where } \|\boldsymbol{v}_i\|_2 = \|\hat{\boldsymbol{v}}_i\|_2 = 1, \tag{6.10}$$

where $\boldsymbol{v}_i$ is the original i^{th} eigenvector, $\hat{\boldsymbol{v}}_i$ is the numerically calculated eigenvector, and sep_i is the separation between Jordan blocks, which is calculated as follows.

Let $\boldsymbol{U}$ be an upper triangular matrix such that $\boldsymbol{AQ} = \boldsymbol{QU}$ with $\boldsymbol{Q}$ a unitary matrix ($\boldsymbol{Q}^{-1} = \boldsymbol{Q}^*$). This is the Schur decomposition of $\boldsymbol{A}$. The eigenvalues of $\boldsymbol{A}$ are the diagonal entries of $\boldsymbol{U}$. There are $s!$ different matrices $\boldsymbol{U}$ (where s is the number of Jordan blocks) corresponding to all possible permutations of the eigenvalues of $\boldsymbol{A}$ in the diagonal. Let

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{12} \\ \boldsymbol{0} & \boldsymbol{U}_{22} \end{bmatrix}, \boldsymbol{U}_{11} \in \mathbb{R}^{m\times m}, \boldsymbol{U}_{22} \in \mathbb{R}^{(n-m)\times(n-m)},$$

such that the eigenvalues of $\boldsymbol{U}_{11}$ are the eigenvalues of the i^{th} Jordan block of $\boldsymbol{A}$, $\boldsymbol{J}_i \in \mathbb{R}^{m\times m}$. The separation sep_i of $\boldsymbol{J}_i$ is the smallest difference between any singular value of $\boldsymbol{U}_{11}$ and those of $\boldsymbol{U}_{22}$ [162]. This value can be obtained by computing the smallest singular value of the Kronecker product [121]

$$\boldsymbol{K} = \boldsymbol{U}_{11} \otimes \boldsymbol{I}_{(n-m),(n-m)} - \boldsymbol{I}_{m,m} \otimes \boldsymbol{U}_{22}.$$

However, this computation is expensive. Moreover, a permutation of the matrix $\boldsymbol{U}$ must be executed for each different eigenvalue of $\boldsymbol{U}$. Hence, we look for a solution that avoids computing these values altogether.

First we will find a lower bound for the separation, which can be obtained by applying [102] to the Kronecker product $\boldsymbol{K}$:

$$\sigma_{min}(\boldsymbol{K}) \geq \left(\frac{p-1}{p}\right)^{\frac{p-1}{2}} \det(\boldsymbol{K}) \min\left(\frac{c_{min}}{\prod_1^p c_j}, \frac{r_{min}}{\prod_1^p r_j}\right), \tag{6.11}$$

$$\text{where } \boldsymbol{K} \in \mathbb{R}^{p\times p} \text{ and } p = m(n-m)$$

n is the dimension of the original matrix, m is the dimension of the current Jordan block, c_j is the 2-norm of the j^{th} column of $\boldsymbol{K}$ and r_j the 2-norm of the j^{th} row (with corresponding minima c_{min}, r_{min}).

Let us first look at the case of matrices with all eigenvalues having algebraic multiplicity of 1 (we'll use the apex i to indicate a partition of $\boldsymbol{U}$ relating to the i^{th} Jordan block). In this case $\boldsymbol{U}^i_{11} = \lambda_i$ and, since $\boldsymbol{K}^i$ is an upper triangular matrix and its determinant is therefore the product of its diagonal entries,

$$det(\boldsymbol{K}^i) = \prod_{i\neq j} (\lambda_j - \lambda_i).$$

We also note that

$$\sum_1^p c_j^2 = \|\boldsymbol{K}^i\|_2^2 \text{ and } \sum_1^p r_j^2 = \|\boldsymbol{K}^i\|_2^2.$$

Using the arithmetic and geometric mean inequality [156] we have

$$\prod_{j=1}^{p} c_j \leq \left(\frac{1}{p}\right)^{\frac{p}{2}} \|\boldsymbol{K^i}\|_2^p \text{ and } \prod_{j=1}^{p} r_j \leq \left(\frac{1}{p}\right)^{\frac{p}{2}} \|\boldsymbol{K^i}\|_2^p,$$

where $\|\boldsymbol{K^i}\|_2 = \|\boldsymbol{U^i_{22}} - \lambda_i \boldsymbol{I}\|_2 \leq \|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2$.

Finally, given that for any matrix $\boldsymbol{U}^i_{11}$ we can select any permutation of $\boldsymbol{U}^i_{22}$ such that the first element of $\boldsymbol{K^i}$ is $\min_{i\neq j}(\lambda_j - \lambda_i)$ and given that $\boldsymbol{K^i}$ is upper triangular, this means that $c_{min} = \min_{i\neq j}(\lambda_j - \lambda_i) \leq r_{min}$.

Going back to Equation (6.11), we have:

$$\sigma_{min}(\boldsymbol{K^i}) \geq \left(\frac{p-1}{p}\right)^{\frac{p-1}{2}} \frac{\prod_{i\neq j}(\lambda_j - \lambda_i)}{\left(\frac{1}{p}\right)^{\frac{p}{2}} \|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2^{p-1}}.$$

This term neither depends on the calculation of $\boldsymbol{K}$, nor on the ordering of $\boldsymbol{U}$.

In the case of a matrix $\boldsymbol{U}$ with algebraic multiplicity strictly greater than one, we should remark that the matrix $\boldsymbol{K}$ has dimension $m(n-m)$. Its determinant is

$$\det(\boldsymbol{K^i}) = \left(\prod_{i\neq j}(\lambda_j - \lambda_i)\right)^m,$$

and its norm is

$$\|\boldsymbol{K^i}\|_2 \leq m\|\boldsymbol{U}^i_{22} - \lambda_i \boldsymbol{I}\|_2 + (n-m)\|\boldsymbol{U}^i_{11} - \lambda_i \boldsymbol{I}\|_2 \leq n\|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2,$$

therefore

$$\sigma_{min}(\boldsymbol{K^i}) \geq \left(\frac{n-1}{n}\right)^{\frac{n-1}{2}} \frac{\left(\prod_{i\neq j}(\lambda_j - \lambda_i)\right)^m}{\left(\frac{1}{n}\right)^{\frac{n}{2}} (n\|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2)^{n-1}}.$$

Replacing for (6.10) and using interval arithmetics we get Equation (6.9). The last part of the equation comes from the need for the cosine to be positive. In practice we want a

much smaller number, so if θ_i is too large, then we can report that the result is imprecise and require the use of higher precision. □

Error bounds for unknown Jordan shapes

As stated earlier, the preceding discussion relies on having selected the correct Jordan shape in the first place (that is, the Jordan shape for the theoretical decomposition without calculation errors), which is in most cases unverifiable. This means that our solution thus far can only be fully sound for diagonalisable matrices (i.e., if the separation of the eigenvalues is larger than the error) or those where the Jordan shape is known a-priori. We therefore propose an additional mechanism to deal with the case of non-diagonalisable matrices with unknown Jordan shapes. In the following, the symbol $\mathbf{1} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$ represents a appropriately-sized matrix with elements all equal to one.

Theorem 10. *Given a numerical decomposition of* $\boldsymbol{A}$, $\hat{\boldsymbol{A}} = \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}$, *the interval matrix*

$$\mathbb{A}'_k = \hat{\mathbb{S}}'\hat{\mathbb{J}}'_k\hat{\mathbb{S}}'^{-1}, \text{ where } \hat{\mathbb{J}}'_k = \hat{\boldsymbol{J}}^k + ((\|\hat{\boldsymbol{J}}\|_1 + [e])^k - \|\hat{\boldsymbol{J}}\|_1^k) \text{ and } \hat{\mathbb{S}}' = \hat{\boldsymbol{S}}(\boldsymbol{I} + [e]\mathbf{1}) \quad (6.12)$$

where $[e] = [-e, e]$ *and* $e = \max(n, \|\hat{\boldsymbol{J}}\|)\|\hat{\boldsymbol{S}}^{-1}\boldsymbol{A}\hat{\boldsymbol{S}} - \hat{\boldsymbol{J}}\|_2$, *is an over-approximation of* $\boldsymbol{A}^k$.

Proof. Let

$$\hat{\boldsymbol{S}}^{-1}\boldsymbol{A}\hat{\boldsymbol{S}} = \hat{\boldsymbol{J}} + \boldsymbol{J}_e \Leftrightarrow \boldsymbol{A} = \hat{\boldsymbol{S}}(\hat{\boldsymbol{J}} + \boldsymbol{J}_e)\hat{\boldsymbol{S}}^{-1},$$

where $\boldsymbol{J}_e = \hat{\boldsymbol{S}}^{-1}\boldsymbol{A}\hat{\boldsymbol{S}} - \hat{\boldsymbol{J}}$ is an error matrix computed from the known quantities on the RHS of the equation. Then

$$\boldsymbol{A}^k = \hat{\boldsymbol{S}}(\hat{\boldsymbol{J}} + \boldsymbol{J}_e)^k\hat{\boldsymbol{S}}^{-1}. \quad (6.13)$$

Let $[e'] = [-\|\boldsymbol{J}_e\|_1, \|\boldsymbol{J}_e\|_1]$ and $\mathbb{J}'_e = [e']\mathbf{1}$, then $\boldsymbol{J}_e \subseteq \mathbb{J}'_e$.

Since each element of $\mathbb{J}'_e$ is $[e']$, each element in $\mathbb{J}'^k_e$ will be $n^{k-1}[e']^k = (n\|\boldsymbol{J}_e\|_1)^{k-1}[e']$, therefore $\mathbb{J}'^k_e = (n\|\boldsymbol{J}_e\|_1)^{k-1}\mathbb{J}'_e$. Similarly, $\mathbb{J}'_e\hat{\boldsymbol{J}}^k\mathbb{J}'_e \subseteq \|\boldsymbol{J}_e\|_1\|\hat{\boldsymbol{J}}\|_1^k\mathbb{J}'_e$.

Let

$$[e] = \max(n, \|\hat{\boldsymbol{J}}\|_1)[e'] \text{ and } \mathbb{J}_e = [e]\mathbf{1},$$

so that

$$(n\|\boldsymbol{J}_e\|_1)^{k-1}\,\mathbb{J}'_e \subseteq ([e])^{k-1}\,\mathbb{J}'_e \text{ and } \|\boldsymbol{J}_e\|_1\|\hat{\boldsymbol{J}}\|_1\,\mathbb{J}'_e \subseteq [e]\mathbb{J}'_e.$$

More generally any matrix multiplication with i elements $\hat{\boldsymbol{J}}$ and j elements $\mathbb{J}'_e$ may be over-approximated by $[e]^{j-1}\|\hat{\boldsymbol{J}}\|_1^i\,\mathbb{J}'_e \subseteq [e]^j\|\hat{\boldsymbol{J}}\|_1^i\mathbf{1}$. From the above properties we expand (6.13) replacing for these values and obtain:

$$\begin{aligned}
&\boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}\left(\hat{\boldsymbol{J}}^k + \sum_{i=0}^{k-1}\binom{k}{i}[e]^{k-i}\|\hat{\boldsymbol{J}}\|_1^i\mathbf{1}\right)\hat{\boldsymbol{S}}^{-1} \qquad (6.14)\\
\Rightarrow\ &\boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}\left(\hat{\boldsymbol{J}}^k - \|\hat{\boldsymbol{J}}\|_1^k + \sum_{i=0}^{k}\binom{k}{i}[e]^{k-i}\|\hat{\boldsymbol{J}}\|_1^i\mathbf{1}\right)\hat{\boldsymbol{S}}^{-1}\\
\Rightarrow\ &\boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}\left(\hat{\boldsymbol{J}}^k + ((\|\hat{\boldsymbol{J}}\|_1 + [e])^k - \|\hat{\boldsymbol{J}}\|_1^k)\mathbf{1}\right)\hat{\boldsymbol{S}}^{-1}\\
\Rightarrow\ &\boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}(\boldsymbol{I} + \mathbb{J}_e)\left(\hat{\boldsymbol{J}}^k + ((\|\hat{\boldsymbol{J}}\|_1 + [e])^k - \|\hat{\boldsymbol{J}}\|_1^k)\right)\hat{\boldsymbol{S}}^{-1}.
\end{aligned}$$

□ □

Notice that, since the formula depends on the horizon k, we cannot in general prove soundness for an unbounded time horizon in this case. In the instance of converging models (as is often the case for industrial systems), we may pick a k that practically reaches a fix point in finite time, thus extending the proof for an infinite time horizon.

6.1.3 Interval simplex

The key implementation required for the Algorithm (step 10 in Algorithm 3) is a variant of simplex that can handle interval representations. We first remark that throughout this paper we are looking at over-approximations of desired quantities in order to ensure soundness, and to optimise algorithmic performance. Let us begin by exploring the meaning of a polyhedral description using interval inequalities. An interval polyhedron $[P] = \{\boldsymbol{x} \mid \mathbb{A}\boldsymbol{x} \leq \mathbb{b}\}$ is a union of polyhedra such that

$$[P] = \bigcup P_i, \text{ where } P_i = \{\boldsymbol{x} \mid \boldsymbol{A}_i\boldsymbol{x} \leq \boldsymbol{b}_i\}, \boldsymbol{A}_i \in \mathbb{A} \wedge \boldsymbol{b}_i \in \mathbb{b}. \qquad (6.15)$$

Note that $[P]$ is not guaranteed to be convex even if all P_i are. We begin by simplifying this description.

Theorem 11. *The polyhedron*

$$\hat{P}_i = \left\{ \boldsymbol{x} \mid \boldsymbol{A}_i \boldsymbol{x} \leq \hat{\boldsymbol{b}} \wedge \boldsymbol{A}_i \in \mathbb{A} \wedge \hat{\boldsymbol{b}}^j = \overline{\boldsymbol{b}^j} \right\}, \tag{6.16}$$

where $\mathbb{b}^j$ is the j^{th} row of $\mathbb{b}$ and $\overline{\boldsymbol{b}^j} = \sup(\mathbb{b}^j)$, is a sound over-approximation of P_i.

Proof. Let $\boldsymbol{r}_i^j$ be a row in $\boldsymbol{A}_i$ with $\rho_{P_i}(\boldsymbol{r}_i^j) = \boldsymbol{b}_i^j$ its corresponding support function. Equations (6.15) and (6.16) state that

$$\boldsymbol{x} \in P_i \Leftrightarrow \forall j,\ \boldsymbol{r}_i^j \boldsymbol{x} \leq \boldsymbol{b}_i^j \ \text{ and } \ \boldsymbol{x} \in \hat{P}_i \Leftrightarrow \forall j,\ \boldsymbol{r}_i^j \boldsymbol{x} \leq \hat{\boldsymbol{b}}^j. \tag{6.17}$$

Since $\forall i,\ \boldsymbol{b}_i^j \leq \overline{\boldsymbol{b}^j}$ and $\overline{\boldsymbol{b}^j} = \hat{\boldsymbol{b}}^j$, we have that

$$\boldsymbol{x} \in P_i \Rightarrow \forall j,\ \boldsymbol{r}_i^j \boldsymbol{x} \leq \hat{\boldsymbol{b}}^j \Rightarrow \boldsymbol{x} \in \hat{P}_i \Rightarrow P_i \subseteq \hat{P}_i. \tag{6.18}$$

The above equation shows that $\hat{P}_i$ is an over-approximation of the polyhedron P_i obtained by relaxing the support functions to the upper limit of their intervals. □

Using Theorem 11, we reduce the description of the Abstract Polyhedron to $[P] = \bigcup \hat{P}_i$.

The standard (non-interval) simplex algorithm visits a sequence of contiguous vertices $\boldsymbol{p}_i^p \in P_i$ and computes the *support function* of the last point $\boldsymbol{p}_i^n$ in this sequence in the direction of the objective function $\boldsymbol{v}$ (i.e., $\boldsymbol{v} \cdot \boldsymbol{p}_i^n$). Hence, to develop an interval simplex we need to describe the vertices of $[P]$ in a way that can be traversed and yields a solution $[\boldsymbol{v}] \cdot \mathbb{p}^n$.

Definition 5. *An extreme point of a polyhedron P is any point in the polyhedron touching at least one of its faces. Let $\boldsymbol{A}_i^p$ be a subset of rows of $\boldsymbol{A}_i$ with $\hat{\boldsymbol{b}}^p$ a subset of the*

corresponding rows in $\hat{\boldsymbol{b}}$ *and* $\boldsymbol{A}_i^{/p} \wedge \hat{\boldsymbol{b}}^{/p}$ *their complementary rows and vector elements, respectively. An extreme point* $\hat{\boldsymbol{p}}_i^p$ *is defined by the equation:*

$$\boldsymbol{A}_i^p \hat{\boldsymbol{p}}_i^p = \hat{\boldsymbol{b}}^p \text{ and } \boldsymbol{A}_i^{/p} \hat{\boldsymbol{p}}_i^p \leq \hat{\boldsymbol{b}}^{/p}. \tag{6.19}$$

Definition 6. *A vertex of a polyhedron P is an extreme point in the polyhedron touching as many faces as the dimension of the polyhedron, namely*

$$\boldsymbol{A}_i^p \hat{\boldsymbol{p}}_i^p = \hat{\boldsymbol{b}}^p \wedge \boldsymbol{A}_i^{/p} \hat{\boldsymbol{p}}_i^p < \hat{\boldsymbol{b}}^{/p} \mid \hat{\boldsymbol{p}}_i^p \in \mathbb{R}^n \wedge |p| = n. \tag{6.20}$$

where $|p|$ *represents the number of rows in* $\boldsymbol{A}_i^p$.

Definition 7. *An abstract vertex* $\mathbb{p}^p \in [P]$ *is a hyper-box containing a corresponding vertex for each polyhedron in the collection* $\hat{\boldsymbol{p}}_i^p \in \hat{P}_i$, *so that* $\mathbb{p}^p \supseteq Conv\left(\bigcup_i \hat{\boldsymbol{p}}_i^p\right)$. *In the following we will replace the index p representing the basis of the vertex with the index k representing the order in which vertices are visited. For a visual representation, see Figure 6.1, where each set of half-planes* $\boldsymbol{r}_i^j \boldsymbol{x} \leq \hat{\boldsymbol{b}}^j$ *(sets of lines marked* $j = 1, 2, 3$ *where each line represents the index i) intersects with another at an* abstract vertex $\mathbb{p}^p$ *(boxes). We can find multiple intersections inside each box corresponding to* $\hat{\boldsymbol{p}}_i^p$.

Definition 8. *A basis* $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ *is a set of independent vectors, the linear combination of which spans the space* $\mathbb{R}^n$.

Theorem 12. *Given a pivot operation* $pv\left(\boldsymbol{p}_i^k, \boldsymbol{p}_i^{k+1}\right), \boldsymbol{p}_i^k \rightarrow \boldsymbol{p}_i^{k+1}$, *an abstract pivot is a transformation*

$$pv\left(\mathbb{p}^k, \mathbb{p}^{k+1}\right), \text{ where } \forall i,\ \hat{\boldsymbol{p}}_i^k \in \mathbb{p}^k \rightarrow \hat{\boldsymbol{p}}_i^{k+1} \in \mathbb{p}^{k+1}. \tag{6.21}$$

Notice that the pivot can be performed on any point, thus it is not limited to vertices or points within the polyhedron (this allows our abstract pivot *to take effect on all points in the hyper-box of the abstract vertex).*

Proof. Let $\boldsymbol{B}_i^k$ be a basis for $\hat{P}_i$ related to point $\boldsymbol{p}_i^k$, and such that

$$\boldsymbol{A}_i \boldsymbol{p}_i^k + \boldsymbol{B}_i^k \boldsymbol{s}_i^k = \hat{\boldsymbol{b}}, \tag{6.22}$$

where s_i^k is a set of auxiliary variables [35]. The pivot operation $pv\left(\boldsymbol{p}_i^k, \boldsymbol{p}_i^{k+1}\right)$ will change the basis such that $\boldsymbol{B}_i^{k+1} = \left(\boldsymbol{E}^k\right)^{-1} \boldsymbol{B}_i^k$. We therefore have

$$\begin{aligned} &\boldsymbol{A}_i \boldsymbol{p}_i^k + \boldsymbol{B}_i^k \boldsymbol{s}_i^k = \boldsymbol{A}_i \boldsymbol{p}_i^{k+1} + \boldsymbol{B}_i^{k+1} \boldsymbol{s}_i^{k+1} = \boldsymbol{A}_i \boldsymbol{p}_i^{k+1} + \left(\boldsymbol{E}^k\right)^{-1} \boldsymbol{B}_i^k \boldsymbol{s}_i^{k+1} \\ \Rightarrow &\boldsymbol{B}_i^k \boldsymbol{s}_i^k = \boldsymbol{A}_i \left(\boldsymbol{p}_i^{k+1} - \boldsymbol{p}_i^k\right) + \left(\boldsymbol{E}^k\right)^{-1} \boldsymbol{B}_i^k \boldsymbol{s}_i^{k+1}. \end{aligned} \tag{6.23}$$

The reason for using the inverse of $\boldsymbol{E}^k$ in the last equation is because implementations of the simplex often work on the inverse of the basis and the formula is not commutative using interval arithmetics. Since $\boldsymbol{E}^k$ creates a change between two bases spanning the same space, it is invertible.

Let $\mathbb{B}^k \supseteq \bigcup_i \boldsymbol{B}_i^k$ be an over-approximation of the basis related to $\mathbb{p}^k \in [P]$ (*i.e.* the set of bases relating to each point in $\mathbb{p}^k$). An abstract pivot preserves the over-approximation of the bases in Equations (6.21) and (6.23) since:

$$\forall k, \exists \mathbb{E}^k \supseteq \bigcup_i \boldsymbol{E}_i^k, \text{ where } \mathbb{B}^{k+1} = \left(\mathbb{E}^k\right)^{-1} \mathbb{B}^k \supseteq \bigcup_i \boldsymbol{B}_i^{k+1}. \tag{6.24}$$

Applying interval arithmetics to Equation (6.23) and moving $\mathbb{A}$ to the right, we obtain:

$$\begin{aligned} &\left(\mathbb{E}^k\right)^{-1} \mathbb{B}^k \mathbb{s}^{k+1} \supseteq \mathbb{A}\left(\mathbb{p}^k - \mathbb{p}^{k+1}\right) + \mathbb{B}^k \mathbb{s}^k \\ \Rightarrow &\forall i,\ \left(\mathbb{E}^k\right)^{-1} \boldsymbol{B}_i^k \boldsymbol{s}_i^{k+1} \supseteq \boldsymbol{A}_i \left(\boldsymbol{p}_i^k - \boldsymbol{p}_i^{k+1}\right) + \boldsymbol{B}_i^k \boldsymbol{s}_i^k. \end{aligned} \tag{6.25}$$

Equations (6.24) and (6.25) are satisfiable if we pick large enough intervals for the elements of $\mathbb{E}^k$, thus proving the theorem. □

A new problem arises regarding precision: whereas before we had disjoint vertices $\boldsymbol{p}^k \neq \boldsymbol{p}^{k+1}$, we now have possible intersections $\mathbb{p}^k \cap \mathbb{p}^{k+1} \neq \oslash$. There are three consequences.

First, the over-approximation may become highly imprecise. Second, the algorithm may start cycling between the two intersecting vertices, which may cause the program to not terminate. While imprecision has been defined in Equation (6.1), the question is how to show completeness. We consider the definition of the vertices and Equation (6.3). If $\mathbb{A}\mathbb{p}^k$ is imprecise, then the base $\mathbb{B}^k$ is incomplete, and we abort the simplex, indicating that higher precision is required.

The third effect is that the corresponding confusion between two bases may cause the simplex to pivot erroneously on the second basis (i.e., once a vertex $\mathbb{p}^{k+1}$ is reached, the next pivot may start from $\mathbb{p}^j$, where $\mathbb{p}^j \cap \mathbb{p}^{k+1} \neq \oslash$ and $\exists i$ such that $\boldsymbol{A}_i \boldsymbol{p}_i^j + \boldsymbol{B}_i^{k+1} \boldsymbol{s}_i^{k+1} \neq \hat{\boldsymbol{b}}$). Therefore, before we pick a pivot, we must check that the current Abstract Basis matches the current Abstract Vertex: $\mathbb{A}\mathbb{p}^k + \mathbb{B}^k\mathbb{s}^k - \hat{\boldsymbol{b}} = 0$ (see Equation (6.3)). As with the other two cases, a failed check can be addressed by increasing the precision of the numerical algorithm. If the precision is not allowed to be increased indefinitely (i.e., it has a limit), then the procedure is not complete, since a number of problems (depending on the actual value of the precision) will not terminate with a valid result due to imprecision.

The final stage of the simplex, which corresponds to finding the support function $\max(\mathbb{v} \cdot \mathbb{p}^k)$ is trivially sound since it is the maximum of the resulting interval. Note that as stated at the beginning of this section, this is an over-approximation of the support function, given that $\mathbb{p}^k$ is an over-approximation in itself.

6.1.4 Vertex enumeration

Vertex enumeration (step 9 in Algorithm 3) is an algorithm similar to simplex since it operates on the idea of visiting each vertex once in order to enumerate them.

The standard (non-interval) vertex enumeration algorithm starts by finding a base $\boldsymbol{V}^K$ which contains a number of vertices of the polyhedron. The index K is a set indicating the state of the algorithm by enumerating the rows of $\boldsymbol{A}$ that have been evaluated thus

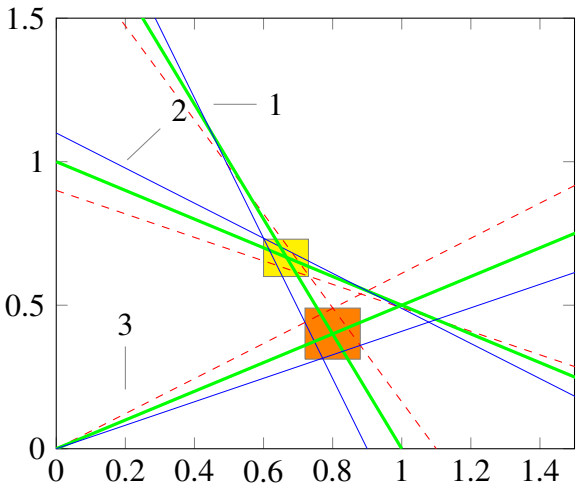

Figure 6.1: Three interval half-planes with negative (dashed red), zero (thick green) and positive (thin blue) angular error representations. The yellow and orange areas (hypercubes) over-approximate all possible vertices of the resulting polyhedron at the given location. If these hypercubes partially intersect, the abstract vertex $\mathbb{p}^k$ must necessarily contain all intersecting hypercubes.

far. The process starts by pivoting over a number of different rows in $\boldsymbol{A}$ (using a simplex algorithm) and by selecting the feasible points visited, which are known to be vertices of the polyhedron. For this stage of the algorithm in the interval case, the use of a simplex as described in Section 6.1.3 ensures overall soundness. The base $\boldsymbol{V}^K$ is then iteratively expanded to $\boldsymbol{V}^{K+j}$ by exploring the j^{th} row of $\boldsymbol{A}$ (denoted $\boldsymbol{r}^j$). The corresponding pairs $(\boldsymbol{A}^{K+j}, \boldsymbol{V}^{K+j})$ are constructed using the information from $(\boldsymbol{A}^K, \boldsymbol{V}^K)$ as follows:

Theorem 13. *Let*

$$\boldsymbol{A}^K \in \mathbb{R}^{n_K \times p} \wedge \boldsymbol{r}_i^j \in \mathbb{R}^{1 \times p} \wedge \boldsymbol{V}^K \in \mathbb{R}^{p \times m_K}.$$

where p is the dimension of the polyhedron, n_K the number of elements in the set K and m_K the number of vertices found up until stage K.

$$H_j^+ = \left\{\boldsymbol{x} \mid \boldsymbol{r}^j\boldsymbol{x} > 0\right\}, \quad H_j^- = \left\{\boldsymbol{x} \mid \boldsymbol{r}^j\boldsymbol{x} < 0\right\}, \quad \textit{and} \quad H_j^0 = \left\{\boldsymbol{x} \mid \boldsymbol{r}^j\boldsymbol{x} = 0\right\},$$

be the spaces outside, inside and on the j^{th} hyperplane with respect to the polyhedron.

and

$$V^{K^+} = \left\{ \boldsymbol{p}^+ \in \boldsymbol{V}^K \cap {H_j}^+ \right\}, \quad V^{K^-} = \left\{ \boldsymbol{p}^- \in \boldsymbol{V}^K \cap {H_j}^- \right\}, \quad \textit{and} \quad V^{K^0} = \left\{ \boldsymbol{p}^0 \in \boldsymbol{V}^K \cap {H_j}^0 \right\}, \tag{6.26}$$

the existing vertex candidates lying in each of these spaces.

New vertex candidates are found as a linear combination of existing ones given a previously unused known constraint $\boldsymbol{r}^j$*:*

$$V^{K+j} = V^K \cup \left\{ \left(\boldsymbol{r}^j \boldsymbol{p}_k^+ \right) \boldsymbol{p}_{k'}^- - \left(\boldsymbol{r}^j \boldsymbol{p}_{k'}^- \right) \boldsymbol{p}_k^+ \mid k \in [1, \dots, m_K^+] \wedge k' \in [1, \dots, m_K^-] \right\}. \tag{6.27}$$

where m_K^- *and* m_K^+ *are the number of vertices in* V^{K^-} *and* V^{K^+} *respectively,* $\boldsymbol{p}_k^-$ *and* $\boldsymbol{p}_k^+$ *are points contained in the sets, and* $\boldsymbol{r}^j$ *is the selected row of* $\boldsymbol{A}$ *to be added to* $\boldsymbol{A}^K$*.*

For the proof see [86].

Let us now consider the interval arithmetics equivalent of this theorem. Interval arithmetics ensure the soundness of the calculation

$$\mathbb{p}_{kk'} \in \boldsymbol{V}^{K+j} = \left(\mathbb{r}^j \mathbb{p}_k^+ \right) \mathbb{p}_{k'}^- - \left(\mathbb{r}^j \mathbb{p}_{k'}^- \right) \mathbb{p}_k^+,$$

so all we need to ensure is the inclusions in (6.26).

If we expand for one of the sets, we get

$$[V^K]^+ = \left\{ \mathbb{p}^+ \in [\boldsymbol{V}^K] \cap [H_j]^+ \mid [H_j]^+ = \left\{ \mathbb{x} \mid \mathbb{r}^j \mathbb{x} > 0 \right\} \right\},$$

where the inclusion in $[H_j]^+$ becomes the concern, namely because using interval arithmetics we may find points that are either partially included (i.e., a portion of the interval, but not all of it, belongs to the set). Once again, we find that Equation (6.3) ensures both the separation of the sets and the correctness of the inclusions.

Benchmark	Dimension	Unsound (ld)	Unsound (mp)	Sound (mpi)	exact
Building	48	$18s$	$185s$	$558s$	*t.o.*
issr10	10	$2s^*$	$23s$	$41s$	*t.o.*
Convoy Car 3	6	$0.3s$	$1.3s$	$3.6s$	$24.6s$
Convoy Car 2	3	$13ms$	$33ms$	$73ms$	$5.46s$
Parabola	4	$12ms$	$12ms$	$47ms$	$2.5s$

Table 6.1: Axelerator: time performance on various benchmarks. Dimension is the number of variables in the problem; ld denotes long double precision; mp is the required precision for the algorithm using non-interval arithmetics; mpi is the sound algorithm; exact is the sound algorithm run using exact arithmetics; t.o. denotes timeout. * returns invalid data (nan)

Theorem 14. *Given the separation criteria in Equation* (6.3)*, the sets* $[V^K]^+$, $[V^K]^-$ *and* $[V^K]^0$ *are disjoint.*

Proof. The proof is direct from the definitions. Any point that may intersect more than one set will be marked as imprecise by (6.3). □

As with the interval simplex, the algorithm is sound but may be incomplete. Since there always exists a precision that implies a sufficiently small rounding error (given that the error decreases monotonically with increasing precision), completeness can be achieved by increasing precision at a higher processing time cost.

6.2 Experimental results

The results discussed in the previous sections have been implemented in the tool Axelerator using *eigen* [95] for algebraic operations and *boost* [150] to manage intervals. Interval comparisons are implemented independently to follow our choice of ordering. The tool has been tested in a number of benchmarks (available with the tool) to determine the nature of the numerical errors. The benchmarks have been first run using unsound standard long double precision and multiple precision arithmetic with the required precision for the problem to be solved correctly (i.e., the precision demanded by our sound algorithm). The results are presented in Table 6.1.

It can be seen from the results that the cost of using sound arithmetics is approximately 3 times that of using floating points of the same precision. The bigger cost for larger dimensional models is the requirement to use a higher precision arithmetic. This happens

because the intervals grow constantly (whereas regular floating point errors often cancel themselves out resulting in a smaller overall error), and the model requires the higher precision to maintain a representative model. Accepting larger errors however can result in both too conservative results and cycling in the simplex (which results in non-termination), so we must accept this need for the algorithm to work. The cost of using an exact arithmetic simplex to evaluate an interval Tableau is combinatorial, hence for example, a 10-dimensional Tableau would require 2^{10} operations which is clearly worse than any time increase required in this paper. The alternative, which is also requiring exact arithmetic in the eigendecomposition, can be very costly (see last column in table 6.1). Thus, our algorithm offers a good trade-off between fast unsound algorithms and slow exact ones.

Chapter 7

Conclusions and Future Work

We have extended abstract acceleration to deal with continuous models with variable inputs, thus making it a strong technique for sound verification of unbounded time continuous LTI dynamics. Furthermore, we have developed numeric techniques to make it scalable while retaining numerical soundness. This places our technique in a privileged position where few tools are able to perform. Our comparison with state of the art tools in reachability analysis of LTI models shows that our implementation is competitive. It is nearly as fast as bounded time reachability tools while retaining precision, and provides much better precision than existing static analysis tools for unbounded loops, performing at similar or faster speeds. In many cases the improvement over existing methods can be measured in orders of magnitude. Finally, we have applied our technique in multiple verification and synthesis scenarios to prove its breadth and usefulness in practical engineering problems.

As mentioned in the related work section, we have yet to extend the implementation to the case of hybrid systems. We can embed abstract acceleration in the Hybrid Automata model to evaluate continuous traces, thus enabling the model to acquire the benefits of this technique while retaining existing techniques for evaluating the discrete transitions. However, because we are not accelerating the crossing of the guards, we fold back to a bounded time model. It is possible in some cases to find a fix-point by discretely evaluating increasingly larger reach tubes after each crossing, but this does not ensure that

we can evaluate unbounded time in all cases. A better approach would be to look for an over-approximation that combines the accelerated dynamics on both sides of a guard by identifying the region that can be reached only after crossing the guard in both directions and the abstract reach tube created by both dynamics (which is yet to be defined). In the case of continuous time systems (and by extension there are means to apply the following to discrete time), we may also look only at the plane in which the guard exists. A mapping of n-dimensional abstract acceleration into a two dimensional plane would allow for the acceleration of the guard to be performed more efficiently to obtain an unbounded time result. It is worth noting that since we are looking for over-approximations, complex guard crossings (i.e. where vertices or edges exist in the guard set), can be found by aggregating the individual behaviour of each guard.

It would also be worth evaluating the benefits of this technique for the case of non-linear systems. An initial approach would be to approximate non-linear dynamics with piecewise affine linear dynamics and evaluate them as a hybrid model as defined above. The interval analysis approach used here allows us to consider intervals for the dynamics of each section of the piecewise model, thus enabling us to use a sound over-approximation of the non-linear dynamics rather than an unsound one. We note however, that large intervals result in very large errors, so a focus on segmentation in order to keep piecewise intervals small would be required. Further scope would be required to apply abstract acceleration directly to non-linear dynamics, where a suitable solution to the acceleration problem is required. Existing work using Berstein polynomials [60] shows promise in finding linear over-approximations to non-linear dynamics and could prove fruitful in offering a solution for abstract acceleration of non-linear models.

Another important task is to optimise the implementation of our simplex algorithm using interval arithmetics, since it was developed based on a rudimentary implementation and therefore it becomes the source of unnecessary slowness when compared to existing bounded time reachability tools (e.g. HYLAA [22]). Modern implementations of the simplex algorithm use heuristics to identify the best choice of pivots. These can have a huge impact on the performance of the algorithm, even in terms of complexity. Hence, a better implementation is likely to improve performance significantly. We note that in both

this work and in [22] a great improvement in performance was obtained by maintaining the state of the simplex for subsequent evaluations. Since the reachability problem requires the evaluation of several objective functions within the same polyhedra, selecting an optimal order in which these are evaluated avoids revisiting vertices, thus allowing to find the solutions to all the objective functions involved in minimal time. In this paper we used a cosine ordering aimed at having the angles between subsequent vectors be as small as possible. This minimises the number of vertices visited when finding the second objective. However ordering by cosine is expensive and it does not provide a globally optimal solution to this problem. Future work should focus on finding optimal orderings that themselves require low computation cost as a trade-off. Another viable option in this regard is to follow the approach taken by [131], using a non-interval simplex to find an approximate solution, and an interval one to obtain the final, sound solution.

Further improvements in reachability analysis could also be achieved by using a combination of techniques. Existing Bounded Model Checking (BMC) algorithms that use unsound floating point approaches could be made sound by using the numeric techniques presented in this work. Furthermore, since abstract acceleration is most efficient when the verification bounds are not very tight a hybrid approach using BMC for a discrete time horizon and abstract acceleration to extend to an infinite time would allow for a fast precise and sound unbounded time solution. This proposition relies on the fact that typically these approaches work on convergent models where the violations would happen towards the beginning of the trace. By using BMC on the initial progressions we ensure maximum precision when it matters, and by completing the trace using a unbounded time algorithm we ensure that there is no risk of us having selected too short of a time horizon. Furthermore, because the abstraction is performed on a reach set that has already converged somewhat, it will be tighter than if we had done so from the initial set.

CEGIS-OR shows promise in exploiting modularity for improved response. It however relies on the use of application-specific optimisation, which requires adequate tools for exploring different types of properties. Our results show significant improvements in terms of scalability over standard CEGIS approaches, with an ability to explore complex properties, hence many synthesis tasks may be solved with it. In particular, we note

that our synthesis loop is more efficient for models where the solution space is formed by connected spaces where optimisation algorithms or localised synthesis are likely to be more efficiently than global synthesis. Finally, our application to control synthesis provides a solid proof of concept for scalable optimal control synthesis of LTI models.

There is still a need to extend CEGIS-OR to MIMO models and to analyse more general properties, which may also require exploring other optimisation methods for control systems. In this case, the first step is extracting separate transfer functions for each combination of inputs and outputs and performing Jury's criteria on each of them. This is achievable using the observable and controllable canonical forms for MIMO models, for which an extra step is required to obtain the transformation matrices $\boldsymbol{T}_c$ and $\boldsymbol{T}_o$. These were not applied in this work for lack of time. Due to the nature of this segmentation, properties such as response and sampling time can be analysed for each block, and optimisation functions equally segmented using these blocks.

Exploration of SMT for CEGIS-OR may result in a more streamlined approach to optimisation based CEGIS. Our current analysis has shown that using floating point arithmetic in SAT solvers for improving plant precision results in a great cost on processing times. Using 32 bit precision floating point, our tool timed out for most models larger than 2 variables (without observer). SMT solvers that combine FWL arithmetics with reals could overcome this limitation. Furthemore, the ability to embed the objective function into an SMT formula could allow for a better overall search pattern.

Overall, we believe we have provided a useful scalable tool that can be applied to existing industrial problems and compete with state of the art techniques while lending itself to easy integration and future development.

Appendix A

Axelerator

The techniques described in the previous chapters have been implemented in a tool called Axelerator. Experiments and comparisons to existing tools were performed using Axelerator.

Our tool implements abstract acceleration using the Eigen[1] algebra package and boost[2] interval types on multiple precision floating point numbers (mpfr)[3].

In the first instance we perform eigen-decomposition of the dynamics. The user will indicate whether they are discrete or continuous dynamics, and may provide a sample time to discretse them, in which case the eigendecomposition is performed on the discretised model. The second stage performs an error analysis to evaluate the maximum error in this decomposition. If the requested data types are sound (*ie* using intervals), the tool over-approximates the dynamics by adding this error. At any stage in the analysis, if a calculation returns an uncertain value due to these intervals, the solution is marked as unsound. In most cases, a sound solution can be found by increasing the numeric precision.

Once the dynamics have been processed, the tool proceeds to calculate the abstract dynamics to be used in the solution. The initial states and inputs are then transformed into a set of verification vectors to be processed by the abstract dynamics. These vectors are

[1] `http://eigen.tuxfamily.org`
[2] `http://www.boost.org/doc/libs/1_63_0/libs/icl/doc/html/index.html`
[3] `http://www.mpfr.org`

grouped into objectives (given by the safe set which is provided as a polyhedra), and if requested, a vector not meeting the specification can then be used as a counterexample to refine the abstract dynamics. This is achieved by finding the iteration for which the vector fails the verification and adding a constraint to the abstract dynamics that minimises the error in the corresponding direction. The underlying algorithm uses an interval simplex that checks for soundness at each iteration (unless unsound arithmetics are used) and either relaxes the solution or tags the problem as unsound when the error grows too large.

A.1 Architecture

Axelerator was developed in c++ using templated packages to allow for the use of different data types in its modelling. The main data type used in this dissertation is multiple precision floating point (mpfr) implemented as mpreal. The tool and dependencies are provided as source code and as such can be run in any platform. The studies presented in this paper were run on 64 bit Ubuntu Linux. The software consists of several modules that provide necessary functionality. These modules are:

1. Auxiliary structures

   (a) RowSort

   This module is an extension to real matrices which provides sorting functions. The algorithm is based on Quicksort. Currently the sorting can be one of the following:

   i. Min. Sorts the matrix by row index.

   ii. Max. Sorts the matrix by reverse row index.

   iii. LexMin. Sorts the matrix rows by minimum column value starting from the left.

   iv. LexMax. Sorts the matrix rows by maximum column value starting from the left.

v. Cos. Sorts the matrix rows by minumum angle (i.e. maximum cosine) to the previous ordered row starting from the first row.

(b) SparseRowSort

Performs RowSort using sparse matrices.

(c) MatToString

This module performs I/O operations, converting matrices and numbers into strings and vice-versa. It follows the formats described in Section A.1.2.

(d) Set

This specialised set class is optimised for memory consumption and speed of intersection and merge functions. It simply states whether and index belongs to the set or not.

(e) ParseOptions

This module will process the command line and input the data into the system (DynamicalSystem or Synthesiser).

2. Polyhedral Abstraction Solver

(a) Tableau

This is the basic simplex tableau containing the basic pivot operations. It contains a feasibility search algorithm which doubles as the basis for optimisation problems. The Tableau uses and inverse basis update during pivots.

(b) DualSimplex

This class implements the dual simplex algorithm (based on revised simplex). Several optimisations have been added such as an ordered search option when solving multiple problems and a counterexample escape sequence used for abstraction refinement.

(c) VertexEnumerator

This class implements the functions required for switching polyhedral representations from inequalities to vertices and viceversa. The algorithm is based

on Komei Fukuda's cdd package [85] implementing the Dual Description Algorithm [86].

(d) Polyhedra

The Polyhedral descriptions and advanced operations are handled by this class. It Uses VertexEnumerator and DualSimplex and may keep either or both representations at any time. The Polyhedra Class implements functions such as Matrix Transforms, Translations, Intersections and Merges.

(e) AdvancedPolyhedra

This Class operates on polyhedral abstractions such as reducing dimensionality of rotating dimensions in order to represent them as radii, tagging inequalities with source identifiers (iterations), and preforming column transformations to match the shape of the eigenspace.

(f) EigenPolyhedra

This class keeps a double representation of a polyhedron in both the problem space and the eigenspace. It also allows access to the circular abstractions.

3. EigenStructure Analysis Modules

(a) JordanSolver

Our core algebra package only provides solutions for diagonalizable matrices. This class extends the numerical eigensolver to provide Jordan forms with geometric multiplicities and their corresponding eigenvalues. Due to the nature of the algorithms, this class operates unsoundly on non-interval data types.

(b) JordanMatrix

This class complements the functionality of the Jordan solver by extending it to interval data types, calculating error bounds and block singular values needed for circular abstractions.

(c) AccelMatrix

This class performs acceleration operations based on the eigenspace for both

regular and circular dimensions. It also provides exponentiated matrix values for other operations.

(d) Abstract Matrix

This class synthesises the Abstract Matrix Polyhedra from its source dynamics. It calculates bounds on pairs of eigenvalues for sets of iterations and aggregates them to define the Abstract Dynamics.

4. System Analysis

(a) DynamicalSystem

This is the main evaluation module. It contains the system description and evaluates reach set and reach tubes for both abstract and iterative algorithms.

(b) DigitalSystem

Performs digitalisation functions in time and space such as sampling and truncation. It further provides functions for detecting and managing required bounds given sampling specifications.

(c) CanonicalSystem

Provides transformation functions for canonical representation (Reachable and Observable forms) as well as optimisation functions for synthesis of controllers and observers.

(d) RefinedDynamicalSystem

This module performs refinement operations. It is responsible for CEGAR loops as well as speed optimisations based on incremental analysis.

(e) Synthesiser

This module implements the synthesis algorithms.

A.1.1 Required packages

Axelerator uses the following opensource packages:

1. Eigen

   The Eigen Library [95] provides a templated algebraic package for basic operations. Amongst the most important, we use matrix addition and multiplication, inverse, eigenspace and singular value decomposition, and linear solvers. The library is very fast and can be used with a wide variety of data types which allows us to extend the representation models at our convenience. Some algebraic formulas such as solving the Sylvester equation and calculating bounds for the eigenvectors and eigenvalues are not provided with the library and had to be developed separately. Mpfr is implemented through a distributed (but currently unsupported) mpreal extension. At the moment of writing, Axelerator was using Eigen 3.3

2. Mpfr

   Since we are dealing with very large scale systems, native data types are often insufficient to deal with the precision we require. We therefore use the multiple precision floating point library mpfr [78]. The library is packaged with the mpreal class which is provided with the Eigen distribution.

3. boost intervals

   We use boost [150] in order to encapsulate numeric interval operations on our calculations. since it is a templated library it also gives us freedom of choosing our desired data types. At the moment of writing, the tool was last tested with boost version 1.59. Intervals in boost implement comparisons in a different manner from our proposed model. For this reason, these operations were reimplemented in the Scalar class to conform to our model. This also presents some restrictions in the use of eigen intrinsic operations. In the case of such an occurrence, we capture the eigen error and recalculate locally using unsound procedures (which tags the solution as such).

4. cbmc

   In the case of program synthesis, our approach uses a combination of Axelerator and CBMC [55]. The latter is the source of our main synthesiser which uses SAT/SMT solvers to find solutions for programs written in C. In our experiments,

we used the default SAT solver minisat [71] and a custom written C program to specify the abstractions produced by Axelerator.

A.1.2 Data formats

Numbers are represented using regular floating point format, where scientific notation is used whenever the magnitude number extends beyond the resolution. Given an [interval] matrix with coefficients

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pp} \end{bmatrix}$$

The text format used by the tool is of the form

$[a_{11}, a_{12}, \cdots, a_{1p}$

$a_{21}, a_{22}, \cdots, a_{2p}$

$\vdots$

$a_{p1}, a_{p2}, \cdots, a_{pp}]$

where the square brackets may be omitted when not part of a larger description. For simplicity we will refer to this format as $< matrix >$. For command line input it is also possible to replace the line feeds for semicolons:

$[a_{11}, a_{12}, \cdots, a_{1p}; a_{21}, a_{22}, \cdots, a_{2p}; \cdots; a_{p1}, a_{p2}, \cdots, a_{pp}]$

For polyhedral sets where

$$\begin{bmatrix} c_{11} & c_{12}, & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ c_{r1} & c_{r2} & \cdots & c_{rp} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} < \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_r \end{bmatrix}$$

The representation is

$$
\begin{aligned}
&[c_{11}\ ,\ c_{12}\ ,\ \cdots\ ,\ c_{1p} < d_1\\
&c_{21}\ ,\ c_{22}\ ,\ \cdots\ ,\ c_{2p} < d_2\\
&\vdots\\
&c_{r1}\ ,\ c_{r2}\ ,\ \cdots\ ,\ c_{rp} < d_r]
\end{aligned}
$$

We will refer to this representation as $< \ poly \ >$ Note that the $<$ can also be replaced with a $>$ for inequalities that have a minimum instead of a maximum. The software will negate both sides and produce a set will all maximums, thus result files will have no $>$ symbols.

$$c_{j1}, c_{j2}, \cdots, c_{jp} > d_j \Leftrightarrow -c_{j1}, -c_{j2}, \cdots, -c_{jp} < -d_j$$

The polyhedral representation may also use semicolons instead of line feeds, which is useful when using the command line. Additionally, when interval arithmetics are selected, result files will be printed using a central value and an error range. Items marked as $c_{ij} + e_{ij}$ indicate an interval $[c_{ij} - e_{ij},\ c_{ij} + e_{ij}]$. Note that if there is no error range ($e_{ij} = 0$) the printed value will be c_{ij}.

A.2 Usage

A.2.1 Command line

There are several command line options in accelerator used to input data and set parameters. Command line data will typically have to be enclosed in quotes because of the use of characters such as commas and semicolon in the representation. Multiple files can be listed as separate options.

1. -params $< \ header \ >$

   Sets the problem parameters such as dimension, input dimension, iterations etc. For a detailed description see the header in the File Format.

2. $-$ < *numeric format* > #

   (a) -ld. Sets long double precision floating points (unsound)

   (b) -ldi. Sets long double precision floating point intervals (sound).

   (c) -mp #. Sets multiple precision floating points (unsound) with the given number of bits for the mantissa.

   (d) -mpi #. Sets multiple precision floating point intervals (sound) with the given number of bits for the mantissa.

3. $-$ < *set description* >< *poly* >

   (a) -guard. Sets the guard of the system (forced truncation)

   (b) -init. Sets the initial state (X_0) of the system.

   (c) -oinit. Sets the initial output of the system ($O_0 \mid \boldsymbol{C}X_0 \subseteq O_0$).

   (d) -inputs. Sets the input description (U).

   (e) -ref. Sets the reference input value for closed loop models.

   (f) -sguard. Sets the safety invariant. Template directions will be overriden, taken along the faces of this polyhedron, and the reach tube will be refined to meet this specification.

   (g) -oguard. Sets the safety invariant on the outputs. State safety is directly back-propagated from this, and template directions calculated from that.

4. $-$ < *transform* >< *matrix* >

   (a) -dynamics. Sets the dynamics ($\boldsymbol{A}$ matrix) of the system.

   (b) -isense. Sets the input sensitivity ($\boldsymbol{B}$ matrix) of the system.

   (c) -osense. Sets the output sensitivity ($\boldsymbol{C}$). When this parameter is not set, the tool presumes all the states to be outputs ($\boldsymbol{C} = \boldsymbol{I}$).

   (d) -iosense. Sets the input-output sensitivity ($\boldsymbol{D}$).

   (e) -control. Sets the state feedback control matrix ($\boldsymbol{K}$) for closed loop models.

(f) -observe. Sets the observer feedback ($\boldsymbol{L}$) for closed loop models with observers.

5. mode

   The mode indicates the desired result from axelerator. It indicates whether we are doing forward or backward reachability or if we are in a CEGIS loop.

   (a) aa-tube. This option is the default, it provides an abstract reach tube for the given specification.

   (b) sq-tube. Same as above but uses rectangular over-approximations over the dynamics as opposed to semi-spherical ones on the inputs.

   (c) tube. Provides an explicitly calculated reach tube for the given iterations using support functions and template directions.

   (d) sets. Provides the reach set for the given iteration.

   (e) init. Synthesises a safe initial space for a given problem.

   (f) input. Synthesises a safe input space for a given problem.

   (g) CEGIS. Finds a suitable controller for a given system.

   (h) observer. Finds a suitable controller and observer for a given system.

6. -templates < *matrix* >

   Sets predefined template directions (each row is a direction). If not set, directions are calculated using logahedral faces based on the 't' option of the parameters. If absent, Axelerator uses octagonal templates.

7. -sample #[:#:#]

   Sets the sampling time in seconds (and fractions) for given continuous dynamics. If a second value is provided, it represents the maximum sample delay time which will be used to calculate the delay error. Finally a third value indicates that the provided sampling values are an interval and the system should find an appropriate sampling time between these values.

8. -speed #

   Indicates the maximum desired eigenvalue of a closed loop model. Verification will flag systems with larger eigenvalues, whereas synthesis will aim at finding controllers which generate dynamics that force smaller eigenvalues than this.

9. -spaceex < *name* >

   Reads a program description from a SpaceEx formatted set of files (name.xml and name.cfg). Currently this does not extract the parameters (p,q, etc) from these files, so they have to be set separately.

10. -inc < *option* >

    Sets incremental simplex. The option indicates the sorting algorithm, which can be:

    (a) min. Ascending column value

    (b) max. Descending column value

    (c) cosine. Minimum angle between directions.

11. − < *representation* >

    (a) -ine. Default representation for polyhedral descriptions using inequalities.

    (b) -norm. Use normalised vectors for the polyhedral descriptions.

    (c) -vert. Use vertices for the polyhedral descriptions.

12. − < *canonical form* >

    (a) -CNF. Indicates that the dynamics supplied are in canonical form (i.e. a controller will presume the system is in reachable canonical form and an observer in observable canonical form). Aelerator will translate to the appropriate system given the desired analysis.

    (b) -CNFP. Matrices are supplied assuming the system is in canonical form, but the tool is required to optimise them during analysis.

    (c) -NNF. Matrices are supplied in normal dynamics, i.e. no canonical form. This is the default behaviour.

A.2.2 File formats

The file format follows the nomenclature of the papers:

$$G \rightarrow \boldsymbol{A}X + \boldsymbol{B}U$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices and G, X and U are polyhedral sets (defined as $\boldsymbol{C}X < \boldsymbol{D}$).

There is a header line with special parameters to set the conditions of the problem. The following parameters can be set:

1. p – Dimension of the state space. This parameter is always required.
2. q – Dimension of the parametric inputs. If not present, the problem is assumed to have no inputs. Parametric inputs are not deterministic but assumed to have the same value throughout the progression.
3. v – Dimension of the control inputs. This parameter will override q if present. Control inputs can take any non-deterministic value at each iteration. Over-approximations of control inputs are in general less precise.
4. s – Number of steps (if missing, the problem is assumed unbounded)
5. l – Logahedral order for the abstract matrix directions. The number of faces of the abstract matrix will be $2^{l-1} * p^2$. The higher the order the more precise the abstraction (also the slower the performance). If not present, the default value is 2.
6. t – Logahedral order for the template directions of the reach tube. The number of faces of the reach tube will be approximately $4 * t * p^2$ If not present, the default value is 0 which corresponds to a hypercube.
7. m – Number of bits in the mantissa representation (only valid for mp problems). If not present, the default is either the precision supplied in the command line or 256 in its absence.

8. e – This header is only present on result files and indicates the eigendecomposition error (used for reference of error sources).

A complete file for a 2 dimensional problem could look like this:

$p = 2, q = 2$

$[g_{11}, g_{12} < h_1$

$g_{21}, g_{22} < h_2]$

$->$

$[a_{11}, a_{12}$

$a_{21}, a_{22}]$

$[x_{11}, x_{12} < y_1$

$x_{21}, x_{22} < y_2$

$x_{31}, x_{32} < y_3]$

$+$

$[b_{11}, b_{12}$

$b_{21}, b_{22}]$

$[u_{11}, u_{12} < v_1$

$u_{21}, u_{22} < v_2$

$u_{31}, u_{32} > v_3]$

The result file will look like this:

p=2,q=2,s=50,l=2,t=0,e=1.5e-23

$[r_{11} + e_{11}, r_{12} + e_{12} < s_1 + es_1$

$r_{21} + e_{21}, r_{22} + e_{22} < s_2 + es_2$

$r_{31} + e_{31}, r_{32} + e_{32} < s_3 + es_3$

$r_{41} + e_{41}, r_{42} + e_{42} < s_4 + es_4]$

Bibliography

[1] A. Abate, I. Bessa, D. Cattaruzza, L. C. Cordeiro, C. David, P. Kesseli, and D. Kroening. Sound and automated synthesis of digital stabilizing controllers for continuous plants. In *Hybrid Systems: Computation and Control (HSCC)*. ACM, 2017.

[2] A. Abate, I. Bessa, D. Cattaruzza, L. C. Cordeiro, C. David, P. Kesseli, D. Kroening, and E. Polgreen. Automated formal synthesis of digital controllers for state-space physical plants. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 462–482, 2017.

[3] C. M. Agulhari, R. C. Oliveira, and P. L. Peres. LMI relaxations for reduced-order robust control of continuous-time uncertain linear systems. *IEEE Transactions on Automatic Control*, 57(6):1532–1537, 2012.

[4] T. Akazaki, I. Hasuo, and K. Suenaga. Input synthesis for sampled data systems by program logic. *arXiv preprint arXiv:1501.06005*, 2015.

[5] M. Althoff. An introduction to cora 2015. In *ARCH@ CPSWeek*, pages 120–151, 2015.

[6] M. Althoff, S. Bak, D. Cattaruzza, X. Chen, G. Frehse, R. Ray, and S. Schupp. ARCH-COMP17 category report: Continuous and hybrid systems with linear continuous dynamics. In *ARCH17. 4th International Workshop on Applied Verification*

of Continuous and Hybrid Systems, collocated with Cyber-Physical Systems Week (CPSWeek) on April 17, 2017 in Pittsburgh, PA, USA, pages 143–159, 2017.

[7] R. Alur, R. Bodik, G. Juniwal, M. M. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 1–8. IEEE, 2013.

[8] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1993.

[9] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in charon. In *International Workshop on Hybrid Systems: Computation and Control*, pages 6–19. Springer, 2000.

[10] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[11] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. Lapack: A portable linear algebra library for high-performance computers. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, pages 2–11. IEEE Computer Society Press, 1990.

[12] É. André, L. Fribourg, U. Kühne, and R. Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM 2012: Formal Methods*, pages 33–36. Springer, 2012.

[13] É. André and U. Kühne. Parametric analysis of hybrid systems using hymitator. In *Proceedings of the Posters & Tool Demos Session of the 9th International Conference on Integrated Formal Methods (iFM 12), Pisa, Italy, CNR and ISTI*, pages 16–19, 2012.

[14] A. Anta, R. Majumdar, I. Saha, and P. Tabuada. Automatic verification of control system implementations. In *EMSOFT*, pages 9–18, 2010.

[15] P. Apkarian. Tuning controllers against multiple design requirements. In *System Theory, Control and Computing (ICSTCC), 2012 16th International Conference on*, pages 1–6. IEEE, 2012.

[16] D. Arzelier, D. Georgia, S. Gumussoy, and D. Henrion. H_2 for HIFOO. *arXiv preprint arXiv:1010.1442*, 2010.

[17] D. Arzelier and D. Peaucelle. An iterative method for mixed H_2/H_∞ synthesis via static output-feedback. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 3, pages 3464–3469. IEEE, 2002.

[18] K. Åström and T. Hägglund. *Advanced PID Control*. ISA-The Instrumentation, Systems, and Automation Society, 2006.

[19] K. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Prentice Hall information and system sciences series. Prentice Hall, 1997.

[20] K. J. Astrom and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, NJ, USA, 2008.

[21] 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.

[22] 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.

[23] M. L. Balinski and R. E. Gomory. A mutual primal-dual simplex method. *Recent advances in mathematical programming*, pages 17–26, 1963.

[24] S. Bennett. A brief history of automatic control. *IEEE Control Systems*, 16(3):17–25, 1996.

[25] J. A. Bergstra and C. Middelburg. Process algebra for hybrid systems. *Theoretical Computer Science*, 335(2):215–280, 2005.

[26] I. Bessa, H. Ismail, L. Cordeiro, and J. Filho. Verification of fixed-point digital controllers using direct and delta forms realizations. *Design Autom. for Emb. Sys.*, 20(2):95–126, 2016.

[27] I. Bessa, H. Ismail, R. Palhares, L. Cordeiro, and J. E. C. Filho. Formal non-fragile stability verification of digital control systems with uncertainty. *IEEE Transactions on Computers (to appear)*, 2016.

[28] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. *Symbolic model checking without BDDs*. Springer, 1999.

[29] 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.

[30] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of operations Research*, 2(2):103–107, 1977.

[31] 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.

[32] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *HSCC*, LNCS, pages 73–88. Springer, 2000.

[33] O. Bouissou. *Analyse statique par interprétation abstraite de systèmes hybrides*. PhD thesis, École Polytechnique, 2008.

[34] 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.

[35] S. Bradley, A. Hax, and T. Magnanti. Applied mathematical programming. 1977.

[36] M. Brain, C. Tinelli, P. Rümmer, and T. Wahl. An automatable formal semantics for IEEE-754 floating-point arithmetic. In *ARITH*, pages 160–167. IEEE, 2015.

[37] M. S. Branicky, V. S. Borkar, S. Mitter, et al. A unified framework for hybrid control: background, model, and theory. 1994.

[38] J. Burke, D. Henrion, A. Lewis, and M. Overton. HIFOO–a MATLAB package for fixed-order controller design and H8 optimization. In *Fifth IFAC Symposium on Robust Control Design, Toulouse*, 2006.

[39] H. Bustince, M. Galar, B. Bedregal, A. Kolesarova, and R. Mesiar. A new approach to interval-valued choquet integrals and the problem of ordering in interval-valued fuzzy set applications. *IEEE Transactions on Fuzzy systems*, 21(6):1150–1162, 2013.

[40] Y.-Y. Cao, Y.-X. Sun, and J. Lam. Simultaneous stabilization via static output feedback and state feedback. *IEEE Transactions on Automatic Control*, 44(6):1277–1282, 1999.

[41] C. Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907.

[42] 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.

[43] 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`.

[44] 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.

[45] L. Chen, A. Miné, and P. Cousot. A sound floating-point polyhedra abstract domain. In *Asian Symposium on Programming Languages and Systems*, pages 3–18. Springer, 2008.

[46] 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.

[47] G. Chesi. Robust static output feedback controllers via robust stabilizability functions. *IEEE Transactions on Automatic Control*, 59(6):1618–1623, 2014.

[48] A. Chutinan and B. H. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *CDC*, pages 2089–2094. IEEE Computer Society, 1998.

[49] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Parameter synthesis with IC3. In *FMCAD*, pages 165–168, 2013.

[50] A. Cimatti, S. Mover, and S. Tonetta. Efficient scenario verification for hybrid automata. In *CAV*, pages 317–332. Springer, 2011.

[51] A. Cimatti, S. Mover, and S. Tonetta. SMT-based verification of hybrid systems. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

[52] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.

[53] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent system using temporal logic specifications: a practical approach. In *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 117–126. ACM, 1983.

[54] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT press, 1999.

[55] E. M. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, volume 2988, pages 168–176. LNCS, 2004.

[56] M. A. Colón, S. Sankaranarayanan, and H. B. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, pages 420–432. Springer, 2003.

[57] 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.

[58] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *The Journal of Logic Programming*, 13(2-3):103–179, 1992.

[59] T. Dang and T. M. Gawlitza. Template-based unbounded time verification of affine hybrid automata. In *APLAS*, LNCS, pages 34–49. Springer, 2011.

[60] T. Dang and R. Testylier. Reachability analysis for polynomial dynamical systems using the bernstein expansion. *Reliable Computing*, 17(2):128–152, 2012.

[61] G. Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963.

[62] C. David, D. Kroening, and M. Lewis. Using program synthesis for program analysis. In *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, pages 483–498, 2015.

[63] D. C. B. de Oliveira and D. Monniaux. Experiments on the feasibility of using a floating-point simplex in an smt solver. In *PAAR@ IJCAR*, pages 19–28, 2012.

[64] G. Debreu. Representation of a preference ordering by a numerical function. *Decision processes*, 3:159–165, 1954.

[65] 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.

[66] A. Deshpande, A. Göllü, and P. Varaiya. Shift: A formalism and a programming language for dynamic networks of hybrid automata. In *International Hybrid Systems Workshop*, pages 113–133. Springer, 1996.

[67] J. Dong and G.-H. Yang. Robust static output feedback control synthesis for linear continuous systems with polytopic uncertainties. *Automatica*, 49(6):1821–1829, 2013.

[68] A. Donzé, G. Clermont, A. Legay, and C. J. Langmead. Parameter synthesis in nonlinear dynamical systems: Application to systems biology. In *Annual International Conference on Research in Computational Molecular Biology*, pages 155–169. Springer, 2009.

[69] 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.

[70] Y. Ebihara, D. Peaucelle, and D. Arzelier. On the structure of generalized plant convexifying static H8 control problems. *Automatica*, 50(6):1706–1714, 2014.

[71] N. Een and N. Sörensson. Minisat v2. 0 (beta). *Solver description, SAT race*, 2006, 2006.

[72] 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.

[73] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[74] S. Fadali and A. Visioli. *Digital Control Engineering: Analysis and Design*, volume 303 of *Electronics & Electrical*. Elsevier/Academic Press, 2009.

[75] A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *HSCC*, pages 326–341. Springer, 2004.

[76] S. Felsner. *Interval orders: combinatorial structure and algorithms*. Technische Universität Berlin, 1992.

[77] J. Fiala, M. Kočvara, and M. Stingl. PENLAB: A MATLAB solver for nonlinear semidefinite optimization. *arXiv preprint arXiv:1311.5240*, 2013.

[78] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):13, 2007.

[79] B. A. Francis and W. M. Wonham. The internal model principle of control theory. *Automatica*, 12(5):457–465, 1976.

[80] G. Franklin, D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Pearson, 7th edition, 2015.

[81] 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.

[82] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *HSCC*, volume 3414 of *LNCS*, pages 258–273. Springer, 2005.

[83] 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.

[84] G. Frehse, S. K. Jha, and B. H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *International Workshop on Hybrid Systems: Computation and Control*, pages 187–200. Springer, 2008.

[85] K. Fukuda. cdd/cdd+ reference manual. *Institute for Operations Research, ETH-Zentrum*, pages 91–111, 1997.

[86] K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and computer science*, pages 91–111. Springer, 1996.

[87] Z. Gajic, M.-T. Lim, D. Skataric, W.-C. Su, and V. Kecman. *Optimal control: weakly coupled systems and applications*. CRC Press, 2008.

[88] V. Galpin, L. Bortolussi, and J. Hillston. Hype: hybrid modelling by composition of flows. *Formal Aspects of Computing*, pages 1–39, 2013.

[89] 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.

[90] 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.

[91] A. Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.

[92] 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.

[93] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *SAS*, LNCS, pages 144–160. Springer, 2006.

[94] L. Gonnord and P. Schrammel. Abstract acceleration in linear relation analysis. *Science of Computer Programming*, 93(Part B):125–153, 2014.

[95] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[96] 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.

[97] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *CAV*, volume 5123 of *LNCS*, pages 190–203. Springer, 2008.

[98] 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.

[99] T. A. Henzinger. The theory of hybrid automata. *Logic in Computer Science*, pages 278–292, 1996.

[100] 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.

[101] T. A. Henzinger and H. Wong-Toi. Using hytech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications*, pages 265–282. Springer, 1996.

[102] Y. Hong and C.-T. Pan. A lower bound for the smallest singular value. *Linear Algebra and its Applications*, 172:27–32, 1992.

[103] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[104] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic*. Aug. 2008.

[105] H. Ismail, I. Bessa, L. C. Cordeiro, E. B. de Lima Filho, and J. E. C. Filho. DSVerifier: A bounded model checking tool for digital systems. In *Model Checking Software (SPIN)*, volume 9232, pages 126–131. LNCS, 2015.

[106] S. Itzhaky, S. Gulwani, N. Immerman, and M. Sagiv. A simple inductive synthesis methodology and its applications. In *ACM Sigplan Notices*, volume 45, pages 36–46. ACM, 2010.

[107] B. Jeannet. Interproc analyzer for recursive programs with numerical variables, 2010. `http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`.

[108] B. Jeannet, P. Schrammel, and S. Sankaranarayanan. Abstract acceleration of general linear loops. In *POPL*, pages 529–540. ACM, 2014.

[109] S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 215–224. ACM, 2010.

[110] T. T. Johnson and S. Mitra. Passel: A verification tool for parameterized networks of hybrid automata, 2012. `https://publish.illinois.edu/passel-tool/`.

[111] A. Julius, S. Strubbe, and A. van der Schaft. Control of hybrid behavioral automata by interconnection. In *Analysis and Design of Hybrid Systems 2003 (ADHS 03): A Proceedings Volume from the IFAC Conference, St. Malo, Brittany, France, 16-18 June 2003*, page 105. Access Online via Elsevier, 2003.

[112] R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM (JACM)*, 33(4):808–821, 1986.

[113] A. Karimi and M. S. Sadabadi. Fixed-order controller design for state space polytopic systems by convex optimization. *IFAC Proceedings Volumes*, 46(2):683–688, 2013.

[114] G. Klein and B. Moore. Eigenvalue-generalized eigenvector assignment with state feedback. *IEEE Transactions on Automatic Control*, 22(1):140–141, 1977.

[115] C. Knospe. Pid control. *IEEE Control Systems*, 26(1):30–31, 2006.

[116] P. Kokotovic, J. Allemong, J. Winkelman, and J. Chow. Singular perturbation and iterative separation of time scales. *Automatica*, 16(1):23 – 33, 1980.

[117] U. Konigorski. Pole placement by parametric output feedback. *Systems & Control Letters*, 61(2):292–297, 2012.

[118] H. Köroğlu and P. Falcone. New lmi conditions for static output feedback synthesis with multiple performance objectives. In *53rd IEEE Conference on Decision and Control*, pages 866–871. IEEE, 2014.

[119] D. Kroening and O. Strichman. Efficient computation of recurrence diameters. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 2575 of *LNCS*, pages 298–309. Springer, 2003.

[120] P. Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, 2nd edition, 1984.

[121] A. J. Laub. *Matrix analysis for scientists and engineers*. Siam, 2005.

[122] 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.

[123] G. Li. On pole and zero sensitivity of linear systems. *IEEE Trans. on Circuits and Systems–I: Fundamental Theory and Applications*, 44(7):583–590, 1997.

[124] J. Liu and N. Ozay. Finite abstractions with robustness margins for temporal logic-based control synthesis. *Nonlinear Analysis: Hybrid Systems*, 22:1–15, 2016.

[125] R. Löhner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, Universität Karlsruhe, 1988.

[126] N. Lynch, R. Segala, F. Vaandrager, and H. B. Weinberg. *Hybrid i/o automata*. Springer, 1996.

[127] L. Marconi, R. Naldi, and L. Gentili. A control framework for robust practical tracking of hybrid automata. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 661–666. IEEE, 2009.

[128] S. Mayer, R. Dehnert, and B. Tibken. Controller synthesis of multi dimensional, discrete LTI systems based on numerical solutions of linear matrix inequalities. In *2013 American Control Conference*, pages 2386–2391. IEEE, 2013.

[129] M. Mazo, Jr., A. Davitian, and P. Tabuada. PESSOA: A tool for embedded controller synthesis. In *Computer Aided Verification (CAV)*, volume 6174 of *LNCS*, pages 566–569. Springer, 2010.

[130] F. McMorris. Interval orders and interval graphs. a study of partially ordered sets (peter c. fishburn). *SIAM Review*, 29(3):484–486, 1987.

[131] D. Monniaux. On using floating-point computations to help an exact linear arithmetic decision procedure. In *International Conference on Computer Aided Verification*, pages 570–583. Springer, 2009.

[132] R. E. Moore. *Interval analysis*, volume 4. Prentice-Hall Englewood Cliffs, 1966.

[133] A. Morgenstern. Symbolic controller synthesis for ltl specifications. 2010.

[134] A. Nerode and W. Kohn. *Models for hybrid systems: Automata, topologies, controllability, observability*. Springer, 1993.

[135] V. A. Oliveira, E. F. Costa, and J. B. Vargas. Digital implementation of a magnetic suspension control system for laboratory experiments. *IEEE Transactions on Education*, 42(4):315–322, Nov 1999.

[136] 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.

[137] R. Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press, 1955.

[138] Y. J. Peretz. A randomized approximation algorithm for the minimal-norm static-output-feedback problem. *Automatica*, 63:221–234, 2016.

[139] A. Platzer. Differential dynamic logic for verifying parametric hybrid systems. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 216–232. Springer, 2007.

[140] S. Pramod and M. Chidambaram. Closed loop identification of transfer function model for unstable bioreactors for tuning PID controllers. *Bioprocess Engineering*, 22(2):185–188, Feb 2000.

[141] J. Pryce. The forthcoming ieee standard 1788 for interval arithmetic. In *International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics*, pages 23–39. Springer, 2015.

[142] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.

[143] H. Ravanbakhsh and S. Sankaranarayanan. Counter-example guided synthesis of control Lyapunov functions for switched systems. In *Conference on Decision and Control (CDC)*, pages 4232–4239, 2015.

[144] H. Ravanbakhsh and S. Sankaranarayanan. Robust controller synthesis of switched systems using counterexample guided framework. In *EMSOFT*, pages 8:1–8:10. ACM, 2016.

[145] G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Transactions on Automatic Control*, 62(4):1781–1796, 2017.

[146] P. Roux, R. Jobredeaux, and P. Garoche. Closed loop analysis of control command software. In *HSCC*, pages 108–117. ACM, 2015.

[147] Y. Saad. *Numerical methods for large eigenvalue problems*, volume 158. SIAM, 1992.

[148] M. S. Sadabadi and A. Karimi. Fixed-order control of LTI systems subject to polytopic uncertainty via the concept of strictly positive realness. In *2015 American Control Conference (ACC)*, pages 2882–2887. IEEE, 2015.

[149] S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *CAV*, volume 6806 of *LNCS*, pages 686–702. Springer, 2011.

[150] B. Schäling. *The boost C++ libraries*. Boris Schäling, 2011.

[151] R. Schmid, L. Ntogramatzidis, T. Nguyen, and A. Pandey. A unified method for optimal arbitrary pole placement. *Automatica*, 50(8):2150–2154, 2014.

[152] P. Schrammel. Unbounded-time reachability analysis of hybrid systems by abstract acceleration. In *Embedded Software*, pages 51–54. IEEE, 2015.

[153] P. Schrammel and B. Jeannet. Applying abstract acceleration to (co-)Reachability analysis of reactive programs. *Journal of Symbolic Computation*, 47(12):1512–1532, 2012.

[154] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat. Combinatorial sketching for finite programs. In J. P. Shen and M. Martonosi, editors, *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006*, pages 404–415. ACM, 2006.

[155] M. W. Spong. The swing up control problem for the acrobot. *IEEE Control Systems*, 15(1):49–55, Feb 1995.

[156] J. M. Steele. *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities*. Cambridge University Press, 2004.

[157] F. Tadeo, O. P. Lopez, and T. Alvarez. Control of neutralization processes by robust loop shaping. *IEEE Transactions on Control Systems Technology*, 8(2):236–246, Mar 2000.

[158] R. H. G. Tan and L. Y. H. Hoo. DC-DC converter modeling and simulation using state space approach. In *IEEE Conference on Energy Conversion, CENCON*, pages 42–47, Oct 2015.

[159] L. Tavernini. Differential automata and their discrete simulators. *Nonlinear analysis*, 11(6):665–683, 1987.

[160] K. H. TC Chen, CY Chang. Reduction of transfer functions by the stability-equation method. *Journal of the Franklin Institute*, 308(4):389 – 404, 1979.

[161] H.-D. Tran, L. V. Nguyen, and T. T. Johnson. Large-scale linear systems from order-reduction (benchmark proposal). In *3rd Applied Verification for Continuous and Hybrid Systems Workshop (ARCH), Vienna, Austria*, 2016.

[162] C. F. Van Loan. Matrix computations (johns hopkins studies in mathematical sciences), 1996.

[163] H. M. Wagner. A comparison of the original and revised simplex methods. *Operations Research*, 5(3):361–369, 1957.

[164] T. E. Wang, P. Garoche, P. Roux, R. Jobredeaux, and E. Feron. Formal analysis of robustness at model and code level. In *HSCC*, pages 125–134. ACM, 2016.

[165] M. Zamani, M. Mazo, and A. Abate. Finite abstractions of networked control systems. In *IEEE CDC*, pages 95–100, 2014.