Verification of advanced controllers for safety-critical systems

Kestutis Siaulys

Department of Engineering

University of Cambridge

This thesis is submitted for the degree of Doctor of Philosophy.

St Catharine's College June 2017

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Kestutis Siaulys
June 2017

Abstract

Verification of advanced controllers for safety-critical systems

Kestutis Siaulys

In order to design and deploy a feedback controller in a real application, one must determine suitable specifications that the design must meet ("validate"), and then ensure that the chosen specifications have been met ("verify").

In this thesis, we investigate a verification paradigm based on formal methods, such as the Satisfiability Modulo Theories (SMT) and quantifier elimination (Weispfenning's virtual term substitution and quantifier elimination by cylindrical algebraic decomposition) algorithms. Any control design requirement (such as satisfactory performance, robustness to uncertainties, stability, etc.) that can be expressed in a first order logic formula can be (in principle) verified by using one of these methods.

Consequently, in principle, this allows us to consider problems like general non-convex optimisation, exact computation of structured singular value, and synthesis of non-convex feasible parameter sets. In practice, the generality of algorithms like quantifier elimination by cylindrical algebraic decomposition come with a downside of high running time when applied to more complex systems with more parameters. This, in some cases, limits the complexity of the system that we could consider.

Therefore, we focused our attention on control problems such as obtaining an explicit MPC law for a linear time invariant system with a quadratic objective and polytopic constraints, or computation of the structured singular value for a system under parametric (and not norm-bounded) uncertainty. Such problems can be expressed as quantifier elimination problems with a particular quantification structure that allows us to take advantage of a specialised

quantifier elimination algorithm — the quantifier elimination by Weispfenning's virtual term substitution procedure that has much lower worst-case running time on these types of problems than quantifier elimination by cylindrical algebraic decomposition algorithm.

Despite these constraints, we were able to apply a quantifier-elimination-based verification framework to clearance of a flight control law developed for a real world industrial system from the aerospace field not only at particular combination of parameters but throughout the whole flight envelope.

In conclusion, while in principle formal methods are applicable to a large body of problems arising in control theory, more widespread practical application depends on further research in efficiency and running time improvement in the implementation of these algorithms.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Scope of Work

Before an advanced feedback control law, such as a Model Predictive Control (MPC) based controller that contains an optimisation algorithm inside the loop, can be implemented and deployed in a safety critical system such as fault-tolerant flight control, it has to be extensively validated and verified in order to ensure that required specifications are met. *Validation* of the system deals with determining the specifications (such as satisfactory performance, robustness to uncertainties, stability, etc.) that the system must meet, while *verification* is concerned with checking that the system satisfies this chosen list of specifications. By definition, validation is a subjective process that relies on making judgements of what real world requirements are important. On the other hand, verification is a more objective step — once the required specifications are expressed mathematically, they can be checked rigorously. Hence, throughout this thesis, we focus our attention on clearly defined verification criteria that can be expressed precisely in mathematical form.

In this thesis, we introduced novel approach to verification based on formal methods (Satisfiability Modulo Theory (SMT) solvers, quantifier elimination (QE) algorithms such as Weispfenning's virtual term substitution and quantifier elimination by cylindrical algebraic decomposition (CAD)). In a nutshell, this verification approach works by expressing a verification criterion of choice in a mathematical form that one of the SMT solvers or QE algorithms is (in principle) capable of checking (subject to decidability).

Consequently, this allows us to analyse the problem from a different angle and, in some cases, obtain results that standard control analysis techniques are not necessarily capable of. This is the case because formal methods, essentially, work by algebraically manipulating the mathematical expression representing the verification criterion to arrive at the conclusion and do not involve numerical techniques such as simulation or gridding. This guarantees that verification by formal methods does not miss a critical frequency or parameter combination at which the desired property is violated. Additionally, the generality of these methods and, in particular, of the CAD-based QE algorithm, allows us (in principle) to relax conditions such as convexity of the cost function in the optimisation problem. This enables us to express general optimisation, computation of structured singular value μ and Linear Temporal Logic (LTL) specification problems as quantifier elimination problems, and to compute some results that standard techniques are not capable of — for example, synthesising feasible parameter sets for nonlinearly parametrized systems in the LTL case.

The downside to the proposed quantifier-elimination-based approach that relies on a general QE algorithm (such as quantifier elimination by cylindrical algebraic decomposition) is the computational penalty that has to be paid when compared to numerical methods. This severely limits the complexity of systems that can be verified using this kind of QE algorithm. Therefore, we limit our attention to control problems that can be expressed as equivalent quantifier elimination ones with a particular quantification structure where all quantified variables appear at most quadratically. In this case, a specialised QE algorithm (quantifier elimination by Weispfenning's virtual term substitution procedure) that has much lower worst-case running time on these types of formulas, is applicable. The introduction of Weispfenning's algorithm for the solution of control-theoretic problems is a principal contribution of this thesis.

Therefore, instead of attempting to obtain an an explicit MPC law for a general nonlinear time invariant system with a polynomial objective and polynomial constraints, we restrict our attention to a linear time invariant system with a quadratic objective and polytopic constraints. This guarantees that the quantification structure of the resulting quantifier elimination problem is such that the efficient Weispfenning's QE algorithm is applicable. This is also the case if we consider an MPC problem with a quadratic objective and linear constraints but with an underlying system dynamics that are linear in the input but nonlinear in the state (with polynomial state evolution functions).

Similarly, instead of trying to compute the structured singular value μ for a system under a general norm-bounded uncertainty (which results in a computationally intractable problem that requires the use of the CAD-based QE algorithm), we consider a system under structured parametric uncertainty. In this case, all the quantified variables in the equivalent quantifier elimination problem appear linearly, and therefore Weispfenning's quantifier elimination algorithm is again applicable. This allows us to analyse the robust stability of systems subject to three or four parametric uncertainties. In particular, these examples illustrate the important advantage of the quantifier-elimination-based verification framework compared to standard iterative branch and bound methods for computing the structured singular value μ for the system containing only real parametric uncertainties — quantifier elimination based approach computes the exact value of μ in form of an algebraic expression rather than lower and upper numerical bounds on it.

Finally, despite the rather high complexity of the system, we were able to use a quantifier-elimination-based verification framework to verify a control law developed for a real world industrial system from the aerospace field. The control scheme in question was developed for the European 7th Framework project titled "REconfiguration of CONtrol in Flight for Integral Global Upset Recovery (RECONFIGURE)". The aim of this project was to develop advanced aircraft guidance and control techniques that facilitate the automated handling of abnormal events while simultaneously optimizing aircraft performance. In particular, we investigate the performance of a robust vertical-load-factor-tracking control law which is based on application of theory from robust MPC and $\mathscr{H}_2$ control and which is supposed to be used as a backup to the baseline Airbus control law in the case when calibrated airspeed measurement is lost due to multiple simultaneous sensor failures. The verification framework successfully showed for which parts of the flight envelope (in terms of aircraft mass, centre of gravity and altitude) the system (interconnection of the model of the aircraft short period dynamics and the backup controller) performance and robustness conditions of interest hold. This problem was computationally tractable by a quantifier-elimination-based verification framework because it depended on a small number of parameters, and therefore a CAD-based QE algorithm could be utilised. Consequently, this illustrates that, in some particular instances, verification frameworks based on formal methods can be used to verify properties of real world industrial systems.

1.2 Structure of the Thesis

The structure of the thesis is as follows. Chapter 2 provides a review of the relevant literature of formal methods, including SMT solvers, *Why3* and *MetiTarski* tools and QE (Weispfenning's virtual term substitution and quantifier elimination by cylindrical algebraic decomposition) algorithms. Chapter 3 is the main part of the thesis where we consider various verification approaches relying on formal methods that were discussed in the literature review in Chapter 2. In Chapter 3, robustness criteria of interest are defined and translated to the form that formal methods based approaches are capable of verifying. The *Why3* verification framework is used to prove Lyapunov stability of an autonomous system implemented graphically in *Simulink*. Quantifier elimination algorithms are used for analysis of the stability of uncertain systems via calculation of the structured singular value μ, obtaining an explicit MPC solution and verifying properties like recursive feasibility (as presented in Siaulys and Maciejowski (2016)), synthesising feasible parameter sets for various systems and LTL specifications, and checking selected performance and robustness requirements throughout the whole flight envelope for the backup flight control scheme developed for the RECONFIGURE project (Maciejowski et al., 2016).

The thesis is concluded in Chapter 4 which provides a summary of the work done, and some comments on what has been achieved and on the prospects for further progress.

Chapter 2

Literature Review of Formal Methods

2.1 Satisfiability Modulo Theories

2.1.1 Preliminaries

In this section, we give definitions of constructs that will be used throughout the chapter.

A *literal* l is either a *propositional variable* p or its negation $\neg p$. A *clause* C is a disjunction of literals $C = l_1 \vee l_2 \vee \cdots \vee l_n$. Collection P of propositional formulas ϕ is the smallest class with the properties:

- each literal $l \in P$
- if $\phi_0 \in P$, then $\neg\phi_0 \in P$
- if $\phi_0, \phi_1 \in P$, then $\phi_0 \wedge \phi_1 \in P$, $\phi_0 \vee \phi_1 \in P$, $(\phi_0 \Rightarrow \phi_1) \in P$ and $(\phi_0 \Leftrightarrow \phi_1) \in P$.

A *truth assignment* (or *model* M) for a propositional formula ϕ maps propositional variables in ϕ to either *true* or *false*. Truth assignment M *satisfies* ϕ (written as $M \models \phi$) if M makes ϕ evaluate to *true*. Formula ϕ is *satisfiable* if there exists an M such that $M \models \phi$. Otherwise, formula ϕ is *unsatisfiable*. Formula ϕ is called *valid* if for all models M, $M \models \phi$. ϕ_1 and ϕ_2 are *equisatisfiable* if ϕ_1 is satisfiable if and only if ϕ_2 is satisfiable.

A propositional formula ϕ is in a *conjunctive normal form (CNF)* if it is a conjunction of clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, which can equivalently can be expressed as a set of clauses $\{C_1, C_2, \ldots, C_m\}$. Any propositional formula can be converted to CNF in a polynomial time by replacing each compound subformula with a new variable and adding required clauses. For example, consider $\phi = \neg p \vee (q \wedge \neg r)$ (which is clearly satisfiable by, say, the model

$M = \{p = false, q = true, r = false\}$). Let $k_1 = q \wedge \neg r$, express it as $k_1 \Leftrightarrow q \wedge \neg r$, which is equivalent to

$$(\neg k_1 \vee (q \wedge \neg r)) \wedge (\neg q \vee r \vee k_1) = (\neg k_1 \vee q) \wedge (\neg k_1 \vee \neg r) \wedge (\neg q \vee r \vee k_1),$$

which can then be expressed as a set of clauses $K_1 = \{\neg k_1 \vee q,\ \neg k_1 \vee \neg r,\ \neg q \vee r \vee k_1\}$. Then, analogously, let $\phi = \neg p \vee k_1 = k_2$, express it as $k_2 \Leftrightarrow \neg p \vee k_1$, which is equivalent to

$$(\neg k_2 \vee (\neg p \vee k_1)) \wedge ((p \wedge \neg k_1) \vee k_2) = (\neg k_2 \vee \neg p \vee k_1) \wedge (p \vee k_2) \wedge (\neg k_1 \vee k_2),$$

which can then be written as a set of clauses $K_2 = \{\neg k_2 \vee \neg p \vee k_1,\ p \vee k_2,\ \neg k_1 \vee k_2\}$. Hence, the propositional formula ϕ is equisatisfiable to the set of clauses $\{k_2\} \cup K_1 \cup K_2$. From now on, all propositional formulas ϕ are assumed to be in a CNF, unless stated otherwise.

A predicate P is a function of one or more variables in some domain of definition X that returns Boolean values, i.e. it is a function of the form:

$$P : X \to \{true, false\}. \tag{2.1}$$

For example, the predicate $P(x,y) \equiv (x = y + 2)$ with the domain of definition being real numbers (i.e., $x, y \in \mathbb{R}$) evaluates to *true* for $x = 2, y = 0$ and evaluates to *false* for $x = 0, y = 2$. Clearly, by definition, any predicate $P(x_1, \ldots, x_n)$ becomes a *proposition* (a statement that is either *true* or *false* but not both) when specific values are substituted for the variables $x_1, \ldots, x_n$.

Another important concept is *quantification*. Let $P(x_1, \ldots, x_n)$ be a predicate with a domain of definition X (i.e, $x_1, \ldots, x_n \in X$). Then *universal quantification* is a proposition of the form:

$$\forall x_1, \ldots, x_n \text{ in } X, P(x_1, \ldots, x_n). \tag{2.2}$$

(2.2) is defined to be *true* if and only if $P(x_1, \ldots, x_n)$ is *true* for every $x_1, \ldots, x_n$ in X. Similarly, an *existential quantification* is a proposition of the form:

$$\exists x_1, \ldots, x_n \text{ in } X, P(x_1, \ldots, x_n), \tag{2.3}$$

which is defined to be *true* if and only if $P(x_1, \ldots, x_n)$ is *true* for at least one combination of the variables $x_1, \ldots, x_n$ in X. A variable x is called *free* if it is not bound by any quantifier $\forall$, $\exists$. For example, x is free in $\exists y : P(x, y)$ but y is not. A *quantifier-free* formula is a formula not containing universal $\forall$ or existential $\exists$ quantifiers. A *sentence* is a formula without free variables.

Many problems of interest require (or are described more intuitively in) an extension of propositional formula called *first order logic* formula. In a propositional formula the atomic formulas (propositional variables p) have no internal structure — they are either *true* or *false*. In the first order logic formula the atomic formulas are *predicates* which are not allowed to have predicates or functions as arguments. Additionally, in the first order logic, quantification is only allowed over the arguments of predicates.

A formula φ_2 is a logical consequence of a formula φ_1 if every model that makes φ_1 *true* also makes φ_2 *true*. A *theory* T is a set of sentences $F_1,\dots,F_n$ (i.e. a set of formulas without free variables) that is closed under logical consequence, i.e., if $F_1,\dots,F_n \models G$, then $G \in T$. Elements of T are called *theorems* which are constructed by selecting a set of sentences called *axioms* and deducing their logical consequences. Given some theory T, it is said that the first order logic formula φ is satisfiable modulo theory T if $T \cup \{\varphi\}$ is satisfiable. A theory T is *decidable* if there is an algorithm that, given a sentence F, determines whether or not $F \in T$. An algorithm D for a decidable theory T that checks whether a formula φ is satisfiable modulo theory T is called a *decision procedure* for theory T.

In order to illustrate some of theses concepts, consider an example of first order logic formula φ in CNF

$$\varphi \equiv \underbrace{(x = y+2)}_{P(x,y)} \wedge \underbrace{(y \geq z-2)}_{Q(y,z)} \wedge \underbrace{(f(x) = f(z))}_{Z(x,z)}, \tag{2.4}$$

where $x,y,z \in \mathbb{R}$, $f(x)$ is some function over the real numbers (i.e., $\forall x_1,x_2 \in \mathbb{R}, x_1 = x_2 \implies f(x_1) = f(x_2)$) and $P(x,y)$, $Q(y,z)$ and $Z(x,z)$ are appropriate predicates. In this particular case, the clauses are the predicates $P(x,y)$, $Q(y,z)$ and $Z(x,z)$, all of which are required to evaluate to *true* in order for φ to be *true*. By using the fact that $P(x,y)$ must be *true*, variable y can be eliminated in (2.4):

$$\varphi \equiv (x \geq z) \wedge (f(x) = f(z)). \tag{2.5}$$

It is clear that (2.5) evaluates to *true* if we set $x = z$. Hence, the first order logic formula φ is satisfiable by any model M such that $M = \{x,y,z \in \mathbb{R} : x = y+2 = z\}$.

In order to check satisfiability of the formula (2.4), we implicitly relied on two theories: theory of *Linear Real Arithmetic (LRA)* and theory of *Equality with Uninterpreted Functions*

(EUF). The atomic formulas in LRA are of the form:

$$a_1x_1 + \ldots + a_nx_n \; \rho \; b, \tag{2.6}$$

where $a_1,\ldots,a_n,b \in \mathbb{R}$ are real constants, $x_1,\ldots,x_n \in \mathbb{R}$ are real variables and ρ is a relational operator ($\rho \in \{<,\leq,>,\geq,=,\neq\}$). Notice that multiplication of two real variables $x_1 \cdot x_2, x_1, x_2 \in \mathbb{R}$ is not allowed in atomic formulas of this theory. Decision procedure for this theory is based on the dual simplex algorithm (Dutertre and de Moura, 2006). EUF theory is a first order theory defined via the following axioms

Reflexivity: $\forall x,\ x = x$

Symmetry: $\forall x,y,\ x = y \implies y = x$

Transitivity: $\forall x,y,z,\ x = y \wedge y = z \implies x = z$

Congruence: $\forall x_1,\ldots,x_n,y_1,\ldots,y_n,\ (x_1 = y_1) \wedge \ldots (x_n = y_n) \implies f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n)$

where $f(x_1,\ldots,x_n)$ is an uninterpreted function with only its *arity* (i.e., the number of arguments) known. Decision procedure for this theory is based on the congruence closure algorithm (Nieuwenhuis and Oliveras, 2007)

Finding the model (truth assignment) M for the propositional formula ϕ is called the *Boolean satisfiability problem*, or *SAT*. High level description of the algorithm used by the majority of the SAT solvers is given in Section 2.1.2. Finding the model M for the first order logic formula φ is called the *Satisfiability modulo theory problem*, or *SMT*. SMT solvers rely heavily on SAT solvers in order to perform effective case analysis (i.e. finding which predicates in the first order logic formula φ have to be *true* and which ones have to be *false* in order to make φ *true*). This approach, taken by the majority of SMT, solvers will be discussed in Section 2.1.3.

2.1.2 Boolean satisfiability problem

Most of the SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm (Davis et al., 1962), whose pseudocode is depicted in Algorithm 1. The DPLL algorithm performs a systematic search over the search space that is a full binary tree where each vertex is a propositional variable with two children (except for leaf vertices) representing the potential truth assignment (*true* or *false*) for this variable. Hence, the search space for a propositional formula ϕ with n variables is a tree with 2^n leaves, and each path from the root to the leaf represents a potential truth assignment M. Given a propositional CNF formula ϕ, DPLL-based algorithms use three main operations to search this tree for a truth assignment M that satisfies ϕ:

Algorithm 1 Pseudocode for DPLL algorithm

```
function DPLL($\phi$)
    if $\phi$ is a consistent set of literals $L$ then
        return $true$
                                ▷ detection that formula $\phi$ is satisfiable
    if $\phi$ contains an empty clause $E$ then
        return DPLL($\phi \wedge \neg l$)
                                ▷ backtrack in case a single clause $E$ evaluates to false
                                ▷ i.e. try to proceed with an opposite truth value of $l$
    while there is some unit clause $U$ in $\phi$ do
        unit_propagate($\phi$, $U$)
    while there is some pure literal $p$ in $\phi$ do
        pure_literal_elimination($\phi$, $p$)
    Decision($l$)               ▷ heuristically choose an unassigned propositional variable $l$
    return DPLL($\phi \wedge l$)    ▷ equivalent to assigning $l$ to true
```

- **Decision** (also called *branching* or *case-splitting*, denoted as *Decision(l)* in pseudocode) — an unassigned propositional variable l is chosen heuristically and assigned the value of *true* or *false*.
- **Propagation** — deduces the consequences of this partial truth assignment using various deduction rules, including:
  - **Unit-clause rule** (denoted as *unit_propagate(ϕ, U)* in pseudocode) — if, after the partial truth assignment, a clause U is a *unit clause* (i.e. it is composed of a single unassigned literal l), then U can only be made *true* by assigning the value needed in order to make l *true*. This essentially means that no choice for l is required. Therefore, all clauses containing l (other than the unit clause itself) can be removed, and all literals $\neg l$ can be removed from the remaining clauses (since l can not contribute to making those clauses *true*). This procedure significantly reduces the size of the search space. For example, consider formula ϕ composed of a set of clauses $\{a \vee b, \neg a \vee c, \neg c \vee d, a\}$, where a is a unit clause. Then this can be reduced to an equivalent set of clauses $\{c, \neg c \vee d, a\}$, which further gets reduced to $\{c, d, a\}$. Hence, ϕ is satisfiable with a truth assignment $a = d = c = true$ while the value of b can be chosen arbitrarily.

  - **Pure literal elimination** (denoted as *pure_literal_elimination(ϕ, p)* in pseudocode) — if a propositional variable p occurs with only one polarity in the formula ϕ (i.e. only p or $\neg p$ is present in ϕ, but not both), it is called a *pure literal*. All clauses containing pure literal p can be made *true* by an appropriate truth assignment. Hence, all clauses containing p can be deleted, further reducing the search space.

- **Backtracking** (denoted as *DPLL($\phi \wedge \neg l$)* in pseudocode) — if, after the partial truth assignment and propagation, a particular clause E becomes empty (i.e. all literals of E got assigned to *false*), then formula ϕ evaluates to *false* as well. This situation is called a *conflict*, and E is called a *conflicting clause*. This indicates that some earlier truth assignments were incorrect. Therefore, the algorithm must *backtrack* and try a different truth assignment to the branch. In case when where is nowhere to backtrack, the formula ϕ is unsatisfiable (hence, unsatisfiability of formula ϕ can only be detected after an exhaustive search).

Satisfiability of the formula ϕ is detected when the original set of clauses making up ϕ gets reduced to a consistent set of literals L (i.e. no l and $\neg l$ literals in the set of clauses). Also, by setting the values of literals in L to *true*, we obtain the model M that satisfies ϕ (propositional variables that are in the original formula ϕ but do not appear in the set L are allowed to have an arbitrary truth value in the model M).

Many additional improvements of the described basic DPLL algorithm have been proposed over the years, including:

- Efficient heuristics for choosing literal l in the decision step.

- Efficient data structures and indexing techniques that speed up unit-clause rule application.

- Random restarts.

- Preprocessing techniques.

- Improvements to the basic backtracking algorithm, including conflict-driven clause learning and non-chronological backtracking (backjumping). These improvements describe a backtracking method that, after reaching the conflicting clause E, finds (via *implication graph*) the reason of the conflict (i.e. truth assignment to propositional variables), and remembers it (by adding the negation of the conflicting condition to the set of initial clauses) in order to make sure that the same conflict is not reached again.

Since SAT is NP-complete, there is no known way to solve SAT problem in polynomial time and therefore all solvers for solving SAT problem take exponential time in the worst case. Regardless, SAT solvers are capable of solving many real-world formulas seen in practice with tens of thousands of propositional variables and millions of constrains (i.e. clauses). This allows SAT solvers to be applied in various industrial verification scenarios like model-checking that the circuit design has desirable behaviour for all allowable inputs, or that two differing circuit designs are functionally equivalent.

2.1.3 SMT solvers: combining SAT and theory-specific solvers

In this section, we address state of the art approaches to solving the *Satisfiability Modulo Theory* problem: given a first order logic formula $\varphi(x_1,..,x_n)$ and a decidable combination of theories $T = T_1 \cup \ldots \cup T_m$, is there an assignment to the variables $x_1, \ldots, x_n$ that makes φ satisfiable modulo theory T? Two main ways of answering this question are an *eager approach* and a *lazy approach*.

In an eager approach (also known as *bit-blasting*), a first order logic formula first gets translated to a propositional logic formula (so, for example, a 64-bit integer variable would get represented as 64 boolean variables). Then this propositional formula would be checked for satisfiability with an efficient SAT solver. Advantages of this approach are the ability to use off-the-shelf SAT solvers and taking advantage of a continuous improvement in their performance. Nonetheless, there are fundamental drawbacks — translation to propositional logic causes us to lose the structure of the first order logic formula $\varphi(x_1,..,x_n)$ which then leads to the SAT solver finding it difficult to discover "trivial" facts in that particular theory (like the commutativity $a+b=b+a$ property for linear integer arithmetic). Therefore, this approach is mostly utilised for bit-vector arithmetic.

This led to the development of the so-called *lazy approach*, in which a DPLL-based SAT solver (as described in Section 2.1.2) gets integrated together with theory-specific decision procedures for theories $T_1, \ldots, T_m$. The algorithm for this approach usually gets denoted *DPLL(T)*, and the pseudocode for an "offline lazy" version of this approach is depicted in Algorithm 2. With this approach, the first order formula φ gets converted to a propositional formula F in CNF ($F = CNF\_bool(\varphi)$). Then, satisfiability of this propositional formula F gets checked with a DPLL-based SAT solver ($DPLL(F)$). If F is unsatisfiable, so is the first order logic formula φ. Otherwise, the truth assignment M gets produced which gets converted to a theory-specific model $M^{'}$ ($To\_Theory\_T(M)$). Then satisfiability of the model $M^{'}$ gets checked with a theory-specific decision procedure ($theory\_T(M^{'})$). If it returns the status of $M^{'}$ as satisfiable, then we are done. Otherwise, we add the negation of the truth assignment M (which is called a *theory lemma*) to the original propositional formula F in order to make sure that we do not arrive at the same conflict later in the search ($F = F \wedge \neg M$) — this process is called *clause learning*.

Algorithm 2 Pseudocode for DPLL(T) algorithm

```
function LAZY_SMT($\varphi$)
    F = CNF_bool($\varphi$)
    while true do
        [res, $M$] = DPLL(F)
        if res == true then
            $M^{'}$ = To_Theory_T($M$)
            res = Theory_T($M^{'}$)
            if res == true then
                return satisfiable
            else
                $F = F \wedge \neg M$
        else
            return unsatisfiable
```

Such an ‘offline lazy” approach would work in principle, but not in practice. Significant improvements in performance can be achieved by having a tighter integration between an SAT solver and theory-specific solvers $T_i, i = 1, \ldots, m$. Such an approach is called an *online lazy approach*, and it has the following main differences compared to the offline one:

- Theory-driven *backjumping* and *learning*. When *unsatisfiable* is returned by a theory solver, it can produce a conflicting set (i.e. inconsistent subset of the input constraints). Negation of this conflicting set is added to the initial set of clauses in order to make sure that the prover does not reach the same inconsistency again. This then drives *non-chronological backtracking* (i.e. allowing to backtrack over several decisions at once rather than one), consequently reducing the size of the search space.

- *Early pruning* — invoking the theory solver on intermediate truth assignments provided by the SAT solver (i.e., not waiting for SAT solver to find the full truth model M). If theory solver returns *unsatisfiable*, then the algorithm can backtrack immediately, consequently significantly pruning the search tree. One possible disadvantage is that many expensive and possibly unneeded calls to the theory solver may be made. Therefore, some theory solver calling strategy has to be implemented in order to achieve a reasonable trade-off. This may include calling the solver eagerly, i.e. every time a new propositional variable is assigned, calling it after a unit-clause rule application, or calling it based on some heuristic metric.

- Theory solver *incrementality*. This is related to the previous improvement. With early pruning implemented, a set of constraints (i.e. truth assignments to clauses) gets continuously incrementally updated when a new clause is assigned a truth value. Hence, crucially for efficiency (since theory solvers get called very frequently on similar problems), the theory solver must have a property of incrementality — when a new constraint (truth assignment to a clause) is added, there is no need for the theorem prover to redo all theory-specific calculations from scratch. Analogously, it is also crucial for efficiency that the theory solver possesses a property of *backtrackability*, i.e. ability to inexpensively restore its internal state after a constraint (truth assignment to a clause) is removed from the constraint set.

- *Theory propagation*. In case an early pruning check returns that the partial truth assignment M is satisfiable, the theory solver may also return a truth assignment to the unassigned clauses.

2.2 *Why3*: an interface to SMT solvers

Why3 is a tool that will be used to create logic theories that fully represent control systems expressed graphically as *Simulink* (The MathWorks Inc., 2012b) diagrams (analogously to the approach presented in Araiza-Illan et al. (2014)). The *Why3* logic language is a first-order language with *polymorphic types*, extended with *recursive algebraic data types*, *inductive predicates*, *recursive functions* and *predicate symbols* (for more details, see Bobot et al. (2011)).

Once a *Why3* theory representing a *Simulink* diagram is created, it can be sent as input to one of over 20 different automated theorem provers and their Satisfiability Modulo Theory (SMT) solvers. An SMT solver calculates satisfiability of a verification goal in the input theory by using a library of given mathematical definitions, axioms and theories. For our purposes, in Section 3.1.2, we will be using *Why3* to interface with Alt-Ergo, CVC3 and Z3 provers in order to verify the asymptotic stability of a system.

Section 2.2.1 provides a high-level description of *Why3* syntax.

2.2.1 *Why3* syntax

In order to create logical *Why3* theories representing *Simulink* diagrams, the following *Why3* language components will be used:

- **Theories:** all *Why3* logic expressions (such as types, predicates, functions, axioms, lemmas and verification goals) that will be used to describe a *Simulink* diagram must be declared inside a theory, which, in *Why3* syntax, is expressed as:

  ```
  theory <Name_of_theory>
     \\Comment: logic expressions used to describe a Simulink diagram
  end
  ```

  More complicated theories can be constructed modularly by using or cloning basic theories. The `use` statement:

  ```
  use import <file_name>.<Theory_A>
  ```

  just copies the theory `Theory_A` with the same symbols as the ones used in the declaration of `Theory_A`. The `clone` statement:

  ```
  clone <file_name>.<Name_of_Theory> as <New_name_of_Theory>
  with function <name> = <new_name>
  ```

  constructs a local copy of the cloned theory with no reuse of symbols, possibly instantiating some of its abstract (i.e. declared but not defined) symbols. Cloning a theory allows us to create several instances of the same theory with different parameter values (unlike using a theory, which just copies it once). *Why3* has a standard library of some basic theories, such as integers, Boolean and real numbers. For the *Why3* standard library theory `RealInfix` for real numbers (and other *Why3* standard library theories it depends on), see Appendix A.1 on page 137.

- **Functions:** operations on data. Can be polymorphic (i.e. accept different types) and recursive. Functions can be uninterpreted (no explicit definition of the calculation performed by the function) and interpreted (calculation performed by the function is defined explicitly). For our purposes, we will be using uninterpreted functions:

  ```
  function <name> <input_types>  : <output_types>
  ```

  and we will define the action performed by the function via axioms. An example of this approach is provided on page 15 for uninterpreted functions `in1` and `out1` which represent the input and output signals of the 'Gain' *Simulink* block.

- **Lemmas and axioms:** logic statements used to help to prove verification goals:

```
lemma/axiom <Name>: <logic statement>
```

Axioms are statements that are taken to be true without proof and hence can be used directly by SMT solvers to help with a proof of a verification goal. Lemmas have to be proven before being used in the same fashion.

- **Verification goals:** logic expressions which have to be proven by SMT solvers in order to verify that properties of interest hold in the system:

```
goal <Name>: <logic expression>
```

A library of theories for various scalar *Simulink* blocks has been developed by us (for the full list, see Appendix A.2 on page 141). For example, consider the theory developed for a scalar 'Gain' block:

```
theory Gain
   use import int.Int
   use import real.RealInfix

   function in1 int : real
   function out1 int : real
   constant gain : real

   axiom a1 : forall k:int. out1 k = in1 k *. gain
   axiom a2 : forall k:int. in1 k >. 0.0 /\ gain >. 0.0 -> out1 k >. 0.0
   axiom a3 : forall k:int. in1 k <. 0.0 /\ gain <. 0.0 -> out1 k >. 0.0
   axiom a4 : forall k:int. in1 k <. 0.0 /\ gain >. 0.0 -> out1 k <. 0.0
   axiom a5 : forall k:int. in1 k >. 0.0 /\ gain <. 0.0 -> out1 k <. 0.0
end
```

In this theory, the standard *Why3* theories of integers `Int` and real numbers `RealInfix` are imported first in order to allow SMT solvers to argue about statements containing those types of numbers. Two uninterpreted functions, `in1` and `out1`, represent the input and output signals of the 'Gain' block, respectively. Both of these functions take a non-negative integer time step k as input (as exemplified by the `int` keyword in the declaration of the functions `in1` and `out1`) and produce a real number as output (as exemplified by the `real` keyword in the declaration of the functions `in1` and `out1`). The output of the function `in1` represents the value of the input signal of the 'Gain' block at time step k, while output of the function `out1` represents the value of the output signal of the same block at time step k. Constant `gain` represents the scalar gain value of the 'Gain' block. Axiom `a1` represents functionality of the 'Gain' block (i.e. at all time steps k, the value of the output signal of the

’Gain’ block is equal to the value of the input signal multiplied by the value of the gain), while the rest of the axioms (`a2`, `a3`, `a4` and `a5`) define the sign properties of the output signal `out1`.

2.3 Quantifier elimination algorithms

2.3.1 Introduction

Many analysis and synthesis problems in control theory can be represented by the first order formula

$$Q_1x_1 \dots Q_nx_n\varphi(p_1,\dots,p_m,x_1,\dots,x_n), \tag{2.7}$$

where $Q_i \in \{\forall, \exists\}$ are either universal or existential quantifiers and φ is a quantifier-free formula constructed by conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) of atomic formulas of the form $f\,\rho\,0$ (where $f \in \mathbb{R}[p_1,\dots,p_m,x_1,\dots,x_n]$ is a polynomial and $\rho \in \{=,\neq,<,\leq\}$ is a relational operator).

One of the ways to find the values of the parameters $p_1,\dots,p_m$ for which the property (2.7) holds is to feed it as an input formula to a quantifier elimination algorithm that outputs a quantifier-free formula $\Phi(p_1,\dots,p_m)$ such that:

$$\Phi(p_1,\dots,p_m) \equiv Q_1x_1 \dots Q_nx_n\varphi(p_1,\dots,p_m,x_1,\dots,x_n). \tag{2.8}$$

For example, feeding the first-order formula requiring the quadratic to be always positive

$$\Phi(a,b,c) \equiv \forall x\; ax^2+bx+c>0 \tag{2.9}$$

to one of those algorithms results in an output that is an equivalent quantifier-free expression in unquantified parameters a,b,c describing where (2.9) holds:

$$\Phi(a,b,c) = c>0 \wedge \left(\left(b<0 \wedge a>\frac{b^2}{4c}\right) \vee (b=0 \wedge a\geq 0) \vee \left(b>0 \wedge a>\frac{b^2}{4c}\right)\right). \tag{2.10}$$

In the following sections, we will discuss the details of the following quantifier elimination algorithms:

- Quantifier elimination by *Cylindrical Algebraic Decomposition (CAD) algorithm* by Collins (1975), discussed in Section 2.3.3, was the first practical quantifier elimination algorithm. It works by dividing $\mathbb{R}^{n+m}$ space into a disjoint set of regions, called *cells*, in each of which all polynomials from a given set are either positive, negative or zero (i.e., have a constant sign). The main advantages of this algorithm are that it is applicable to any input formula of the form (2.7) and that its output $\Phi(p_1,\ldots,p_m)$ is a disjoint union of cells. The main disadvantage of the algorithm is that it is limited to somewhat simple problems because the upper bound on the size of the CAD (i.e., the number of disjoint cells needed to represent this decomposition) grows doubly exponentially with the number of variables $n+m$ (Collins, 1975) in the first order formula (2.7).

- *Weispfenning's virtual substitution algorithms* by (Loos and Weispfenning, 1993; Weispfenning, 1997), discussed in Section 2.3.2. In general, these virtual substitution algorithms are applicable as long as all quantified variables $x_1,\ldots,x_n$ appear at most quadratically in the input formula (2.7). If all the quantified variables in (2.7) appear linearly, virtual substitution algorithms are guaranteed to be able to eliminate all quantified variables from the outset. If some of the quantified variables $x_1,\ldots,x_n$ in (2.7) appear quadratically, some additional constraints on the first order formula (2.7) must be imposed to ensure the capability to eliminate all quantified variables from the outset (these will be discussed in detail in Section 2.3.2). As will be shown in Section 3.4, many MPC problems (starting with a linear one) can be reduced to this type of first order formula. The main advantage of this algorithm is that its worst-case running time on these types of formulas does not depend on the number of free variables $p_1,\ldots,p_m$ (in contrast to CAD). The main disadvantage of this type of algorithm is that it often significantly increases the size of the formula (i.e. the quantifier-free output $\Phi(p_1,\ldots,p_m)$ given by the virtual substitution algorithm is usually a set of regions that is not mutually disjoint) which might make subsequent calculations involving the output of Weispfenning's algorithm complicated.

2.3.2 Weispfenning's virtual term substitution algorithm

As was mentioned in Section 2.3.1, in general, Weispfenning's virtual substitution algorithms (Loos and Weispfenning, 1993; Weispfenning, 1997) are applicable as long as all quantified variables $x_1,\ldots,x_n$ in (2.7) appear at most quadratically. In principle, in order to ensure that these virtual substitution algorithms are capable of eliminating all the quantified variables in (2.7), additional constraints on the quantification structure $Q_1x_1\ldots Q_nx_n$ and the quantifier-free formula $\varphi(p_1,\ldots,p_m,x_1,\ldots,x_n)$ in (2.7) have to be imposed in order to ensure that degrees of all quantified variables $x_1,\ldots,x_n$ never appear with an order higher than quadratic during successive elimination steps.

According to Weispfenning (1997), the kinds of the first order formula (2.7) where complete quantifier elimination by the virtual substitution algorithm can be guaranteed from the outset are:

- The expression (2.7) is linear in all quantified variables except (possibly) one, which appears quadratically and whose quantifier (either $\exists$ or $\forall$) is outermost or second to outermost in the prefix $Q_1x_1 \dots Q_nx_n$ in (2.7).

- All the occurrences of quantified variables that appear quadratically in (2.7) are separated, i.e. no such two variables appear in the same atomic subformula $f\rho 0$ of (2.7). This trivially ensures that degrees of the remaining quantified variables do not increase after an elimination step.

- All the occurrences of quantified variables that appear quadratically in (2.7) are pure (with the possible exception of the quadratically quantified variable with an outermost quantifier out of quadratically quantified variables), i.e. they contain no corresponding linear term. Clearly, in this case, these variables can be replaced by linear variables via simple substitutions.

- Finally, Weispfenning's virtual substitution algorithm described in Weispfenning (1997) is applicable in cases where the quantified variables occur in degrees higher than quadratic, provided that these degrees can be reduced by polynomial factorization to values of at most 2.

The quantifier elimination algorithm, described in (Weispfenning, 1997) (which is an extension of the algorithm described in (Loos and Weispfenning, 1993)), eliminates quantifiers as follows. For the sake of simplicity (since extension to multiple quantified variables, including universal quantification, is trivial), consider a first order formula of the form (2.7) with a single existentially quantified variable x:

$$\exists x \varphi(p_1, \dots, p_m, x). \tag{2.11}$$

Here φ is quantifier-free and constructed by conjunction and disjunction of atomic formulas $f_i\ \rho_i\ 0$ of the form

$$f_i \equiv \big(a_i(p_1, \dots, p_m)x^2 + b_i(p_1, \dots, p_m)x + c_i(p_1, \dots, p_m)\big)\ \rho_i\ 0\ , i \in I,$$

which are at most quadratic in the quantified variable x, with $a_i(p_1,\dots,p_m)$, $b_i(p_1,\dots,p_m)$ and $c_i(p_1,\dots,p_m)$ being polynomials that are independent of x. Denote:

$$\begin{aligned} D_i &= b_i^2 - 4a_ic_i, \\ \alpha_{i-} &= (-b_i - \sqrt{D_i})/(2a_i), \\ \alpha_{i+} &= (-b_i + \sqrt{D_i})/(2a_i). \end{aligned}$$

Let $\varphi[e/x]$ denote a formula that is obtained from φ by substituting e for x. For example, if

$$\varphi(p,x) \equiv \underbrace{(x^2 - x + p \le 0)}_{f_1} \wedge \underbrace{(-x+2 \le 0)}_{f_2}, \tag{2.12}$$

then

$$\varphi[2/x] \equiv (4 - 2 + p \le 0) \wedge (0 \le 0) \equiv (2 + p \le 0).$$

In general, direct substitution of square-root expressions $e_i = \alpha_{i-}$ or $e_i = \alpha_{i+}$ for x in the atomic formula $f_i\ \rho_i\ 0$ would result in an expression of the form

$$f_i[e_i/x] = \frac{x_i + y_i\sqrt{D_i}}{z_i}\ \rho_i\ 0, \tag{2.13}$$

where x_i, y_i and z_i are appropriate polynomials in $p_1,\dots,p_m$. Expression (2.13) contains an undesirable radical. Hence, in order to eliminate the presence of the radical, instead of a direct substitution, (2.13) is replaced by an equivalent expression via 'virtual substitution'. The form of the equivalent expressions depends on the type of relational operator ρ_i in the atomic formula $f_i\ \rho_i\ 0$ as follows:

$$\begin{aligned} f_i[e_i/x] = 0 \equiv &(x_iy_i \le 0) \wedge (x_i^2 - y_i^2D_i = 0), \\ f_i[e_i/x] \neq 0 \equiv &(x_iy_i > 0) \vee (x_i^2 - y_i^2D_i \neq 0), \\ f_i[e_i/x] \le 0 \equiv &((x_iz_i \le 0) \wedge (x_i^2 - y_i^2D_i \ge 0)) \vee \\ &((y_iz_i \le 0) \wedge (x_i^2 - y_i^2D_i \le 0)), \\ f_i[e_i/x] < 0 \equiv &((x_iz_i < 0) \wedge (x_i^2 - y_i^2D_i > 0)) \vee \\ &((y_iz_i \le 0) \wedge (x_iz_i < 0 \vee x_i^2 - y_i^2D_i < 0)). \end{aligned} \tag{2.14}$$

Finally, let I_1, I_2, I_3, I_4 be the set of indices $i \in I$ such that ρ_i is $=, \le, <, \neq$, respectively. Then

$\exists x\varphi(p_1,\dots,p_m,x)$ is equivalent to the following quantifier-free formula

$$\begin{aligned}
&\exists x\ \varphi(p_1,\dots,p_m,x) \equiv \Phi(p_1,\dots,p_m) \equiv \\
\bigvee_{i\in I_1\cup I_2} &((a_i = 0\wedge b_i \neq 0 \wedge \varphi[-c_i b_i^{-1}/x])\vee \\
&(a_i \neq 0 \wedge D_i \geq 0 \wedge (\varphi[\alpha_{i-}/x])\vee \varphi[\alpha_{i+}/x]))\vee \\
\bigvee_{i\in I_3\cup I_4} &((a_i = 0\wedge b_i \neq 0 \wedge \varphi[(-c_i b_i^{-1}+\varepsilon)/x])\vee \\
&(a_i \neq 0 \wedge D_i \geq 0 \wedge (\varphi[(\alpha_{i-}+\varepsilon)/x])\vee \\
&\varphi[(\alpha_{i+}+\varepsilon)/x]))\vee \varphi[-\infty/x],
\end{aligned} \tag{2.15}$$

where ε denotes a positive infinitesimal. Expressions involving infinitesimals ε in (2.15) are replaced by equivalent ones not containing them according to the following set of rules:

$$\begin{aligned}
f_i[(e+\varepsilon)/x] = 0 \equiv &(a_i = 0)\wedge(b_i = 0)\wedge(c_i = 0), \\
f_i[(e+\varepsilon)/x] \neq 0 \equiv &(a_i \neq 0)\vee(b_i \neq 0)\vee(c_i \neq 0), \\
f_i[(e+\varepsilon)/x] \leq 0 \equiv &f_i[(e+\varepsilon)/x] = 0 \vee f_i[(e+\varepsilon)/x] < 0, \\
f_i[(e+\varepsilon)/x] < 0 \equiv &f_i[e/x] < 0 \vee f_i[e/x] = 0 \wedge \\
&\left(\frac{df_i}{dx}[e/x] < 0 \vee \frac{df_i}{dx}[e/x] = 0 \wedge a_i < 0\right).
\end{aligned}$$

Similarly, expressions containing infinity in (2.15) are replaced by the following ones:

$$\begin{aligned}
f_i[-\infty/x] = 0 \equiv &(a_i = 0)\wedge(b_i = 0)\wedge(c_i = 0), \\
f_i[-\infty/x] \neq 0 \equiv &(a_i \neq 0)\vee(b_i \neq 0)\vee(c_i \neq 0), \\
f_i[-\infty/x] \leq 0 \equiv &f_i[-\infty/x] = 0 \vee f_i[-\infty/x] < 0, \\
f_i[-\infty/x] < 0 \equiv &\bigvee_{n=0}^{2}\left((-1)^n F_n < 0 \wedge \bigwedge_{m=n+1}^{2} F_m = 0\right), \\
&\text{where } F_0 = c_i, F_1 = b_i, F_2 = a_i.
\end{aligned}$$

Example of Weispfenning's algorithm application

Consider

$$\exists x\varphi(p,x) \equiv \exists x\underbrace{(x^2 - x + p \leq 0)}_{f_1}\wedge\underbrace{(-x+2\leq 0)}_{f_2}$$

with:

$$\begin{aligned}
&a_1 = 1, b_1 = -1, c_1 = p,\\
&a_2 = 0, b_2 = -1, c_2 = 2,\\
&D_1 = 1-4p,\ \alpha_{\pm} = \frac{1 \pm \sqrt{D_1}}{2}.
\end{aligned}$$

Hence, according to (2.15), performing direct substitution

$$\begin{aligned}
&\exists x \varphi(p,x) \equiv \Phi(p) \equiv\\
&\{(a_1 = 0 \wedge b_1 \neq 0 \wedge \varphi[-c_1 b_1^{-1}/x]) \vee (a_1 \neq 0 \wedge D_1 \geq 0 \wedge (\varphi[\alpha_{1-}/x] \vee \varphi[\alpha_{1+}/x])\} \vee\\
&\{(a_2 = 0 \wedge b_2 \neq 0 \wedge \varphi[-c_2 b_2^{-1}/x]) \vee (a_2 \neq 0 \wedge D_2 \geq 0 \wedge (\varphi[\alpha_{2-}/x] \vee \varphi[\alpha_{2+}/x])\} \vee \varphi[-\infty/x] \equiv\\
&\left(1-4p \geq 0 \wedge \left(\frac{3+\sqrt{D_1}}{2} \leq 0 \vee \frac{3-\sqrt{D_1}}{2} \leq 0\right)\right) \vee \varphi[2/x] \vee \text{false}
\end{aligned}$$

results in an expression containing radicals $\sqrt{D_1}$. In order to eliminate them, recall the relevant virtual substitution rule from (2.14):

$$\begin{aligned}
\frac{x_i + y_i\sqrt{D_i}}{z_i} \leq 0 \equiv &(x_i z_i \leq 0) \wedge (x_i^2 - y_i^2 D_i \geq 0) \vee\\
&(y_i z_i \leq 0) \wedge (x_i^2 - y_i^2 D_i \leq 0).
\end{aligned}$$

Then:

$$\begin{aligned}
\Phi(p) &\equiv \exists x \left((x^2 - x + p \leq 0) \wedge (-x + 2 \leq 0)\right)\\
&\equiv \left(1-4p \geq 0 \wedge \left(\frac{3+\sqrt{D_1}}{2} \leq 0 \vee \frac{3-\sqrt{D_1}}{2} \leq 0\right)\right) \vee \varphi[2/x] \vee \text{false}\\
&\equiv (1-4p \geq 0) \wedge\\
&((6 \leq 0) \wedge (9 - D_1 \geq 0) \vee (2 \leq 0) \wedge (9 - D_1 \leq 0) \vee\\
&(6 \leq 0) \wedge (9 - D_1 \geq 0) \vee (-2 \leq 0) \wedge (9 - D_1 \leq 0)) \vee (2 + p \leq 0)\\
&\equiv (1-4p \geq 0) \wedge (8 + 4p \leq 0) \vee (2 + p \leq 0)\\
&\equiv \boxed{p \leq -2} \blacksquare
\end{aligned}$$

Moreover, the virtual term substitution algorithm has been extended to formulas in which a single quantified variable appears at most cubically (Weispfenning, 1994). In principle, this algorithm can be extended to formulas in which the quantified variable appears with an unbounded degree, by exploiting Thom's Lemma for representation of real roots — for the state of the art, see Kosta and Sturm (2015); Liiva et al. (2014). Unfortunately, no efficient implementation of the algorithm for these cases exists yet.

We will use Weispfenning's virtual term substitution algorithm implementation in *Mathematica* (Wolfram Research Inc., 2016) for the computational examples in Sections 3.4.5, 3.4.6 and 3.4.7.

2.3.3 Quantifier elimination by cylindrical algebraic decomposition algorithm

In this section, we discuss the quantifier elimination by cylindrical algebraic decomposition algorithm (Collins, 1975). Section 2.3.3.1 provides an intuitive example that illustrates the main concepts of the algorithm without relying on its full mathematical machinery. Section 2.3.3.2 provides necessary definitions. Section 2.3.3.3 gives a detailed description of the algorithm that constructs the cylindrical algebraic decomposition (CAD). In Section 2.3.3.4, it is shown how the obtained CAD is used for quantifier elimination. Finally, we provide a less intuitive example in Section 2.3.3.5 that performs quantifier elimination systematically by using the algorithm discussed in this section.

2.3.3.1 Intuitive example

Consider the quantifier elimination problem

$$\Phi(x_1) \equiv \exists x_2 \varphi(x_1,x_2) \equiv \exists x_2 : (x_1^2+x_2^2 \leq 1) \wedge (x_2 = x_1) \tag{2.16}$$

which, essentially, asks us to find for which values of the variable x_1 the straight line $(x_2 = x_1)$ gets inside the unit disk $(x_1^2+x_2^2 \leq 1)$. The solution to this quantifier elimination problem is obvious — it is the interval $-\frac{\sqrt{2}}{2} \leq x_1 \leq \frac{\sqrt{2}}{2}$, as can be seen in Figure 2.1 (a).

It was trivial to solve this problem by inspection. Here we present the outline of a systematic approach (called *quantifier elimination* by *cylindrical algebraic decomposition*) for solving the quantifier elimination problem (2.16) that illustrates the main ideas of this algorithm without relying on its full mathematical machinery.

Let $f_1(x_1,x_2) = x_1^2+x_2^2-1$ and $f_2(x_1,x_2) = x_2-x_1$. The first part of the procedure is to partition the space $\mathbb{R}^2$ into disjoint regions (called *cells*) where polynomials $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$ have a constant sign (positive, negative or identically zero) throughout each cell, and find a *sample point* $s = (s_1,s_2) \in \mathbb{R}^2$ in each one of those cells. Construction of this partition is achieved in three phases:

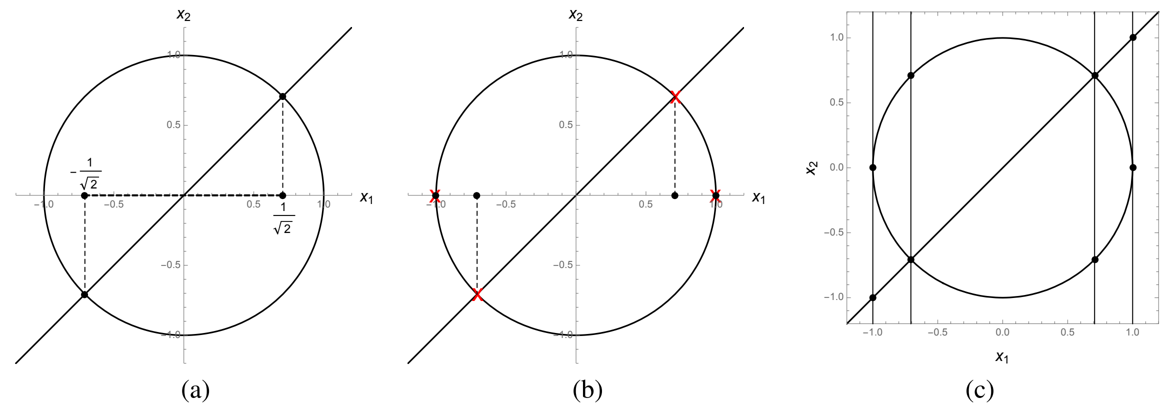


Fig. 2.1 Figure (a) represents the intuitive solution to the quantifier elimination problem (2.16) while figure (b) illustrates the projection operation for the same problem. "Singularities" of the algebraic curves defined by $f_1(x_1,x_2)=0$ and $f_2(x_1,x_2)=0$ are denoted by red crosses while their projections onto the x_1-axis are denoted by black dots. Finally, cylindrical algebraic decomposition of $\mathbb{R}^2$ induced by the polynomials $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$ is shown in figure (c). Dots, line segments between dots and regions of $\mathbb{R}^2$ between those line segments represent disjoint cells $C_i, i=1,\ldots,47$. Throughout each one of those 47 cells, polynomials $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$ have a constant sign.

(i) Find the projections onto the x_1-axis of all the "singularities" (vertical tangents, intersections, isolated points, self-crossings, cusps) of the algebraic curves defined by the polynomials $f_1(x_1,x_2)=0$ and $f_2(x_1,x_2)=0$ (see Figure 2.1 (b)). The projections of these points onto the x_1-axis are

$$\alpha_1=-1,\ \alpha_2=-\frac{\sqrt{2}}{2},\ \alpha_3=\frac{\sqrt{2}}{2},\ \alpha_4=1, \tag{2.17}$$

which decomposes the x_1-axis into 9 components (4 points and 5 open intervals) — see the first column of Table 2.1.

(ii) Now, for each of those 9 segments on the real line, a point belonging to that segment (called a *sample point* s_1), is found. For the projection points $\alpha_i, i=1,\ldots,4$, the sample point s_1 has to be chosen to be α_i, while for the open intervals, any point in that interval can be chosen to be a sample point s_1 (see the second column of the Table 2.1).

Then, both $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$ are evaluated at a sample point $x_1=s_i$ in each of the components, thus giving us $2\times 9=18$ polynomials in x_2 (see the third and fourth columns of the Table 2.1) — essentially, $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$ are evaluated on the

x_1	s_1	$f_1(s_1,x_2)$	$f_2(s_1,x_2)$	Real roots	s_2	$V(C_i)$
$(-\infty,\alpha_1)$	-2	x_2^2+3	x_2+2	-2	$-3,-2,0$	F,F,F
α_1	-1	x_2^2	x_2+1	$-1,0$	$-2,-1,-\frac{1}{2},0,1$	F,F,F,F,F
(α_1,α_2)	$-\frac{4}{5}$	$x_2^2-\frac{9}{25}$	$x_2+\frac{4}{5}$	$-\frac{4}{5},-\frac{3}{5},\frac{3}{5}$	$-1,-\frac{4}{5},-\frac{7}{10},-\frac{3}{5},0,\frac{3}{5},1$	F,F,F,F,F,F,F
α_2	$-\frac{\sqrt{2}}{2}$	$x_2^2-\frac{1}{2}$	$x_2+\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2}$	$-1,-\frac{\sqrt{2}}{2},0,\frac{\sqrt{2}}{2},1$	F,T,F,F,F
(α_2,α_3)	0	x_2^2-1	x_2	$-1,0,1$	$-2,-1,-\frac{1}{2},0,\frac{1}{2},1,2$	F,F,F,T,F,F,F
α_3	$\frac{\sqrt{2}}{2}$	$x_2^2-\frac{1}{2}$	$x_2-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2}$	$-1,-\frac{\sqrt{2}}{2},0,\frac{\sqrt{2}}{2},1$	F,F,F,T,F
(α_3,α_4)	$\frac{4}{5}$	$x_2^2-\frac{9}{25}$	$x_2-\frac{4}{5}$	$-\frac{3}{5},\frac{3}{5},\frac{4}{5}$	$-1,-\frac{3}{5},0,\frac{3}{5},\frac{7}{10},\frac{4}{5},1$	F,F,F,F,F,F,F
α_4	1	x_2^2	x_2-1	$0,1$	$-1,0,\frac{1}{2},1,2$	F,F,F,F,F
$(\alpha_4,+\infty)$	2	x_2^2+3	x_2-2	2	$0,2,3$	F,F,F

Table 2.1 Construction of the partition (called *cylindrical algebraic decomposition*) for the problem represented by $\varphi(x_1,x_2)$.

vertical line rising from the sample point s_1.

(iii) Now, real roots of $f_1(s_1,x_2)=0$ and $f_2(s_1,x_2)=0$ are found for each s_1 (see the fifth column of the Table 2.1), which decomposes the x_2-axis into disjoint components. Then the sample point s_2 is chosen for each one of those components in an analogous manner as before (see the sixth column of the Table 2.1).

Finally, we have obtained 47 sample points $s=(s_1,s_2)$, representing disjoint cells of $\mathbb{R}^2$ where both polynomials $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$ have a constant sign throughout each of those cells. Such a partition is called a *cylindrical algebraic decomposition*, and is depicted in Figure 2.1 (c).

Finally, after the cylindrical algebraic decomposition is constructed, the existential quantifier in (2.16) can be eliminated as follows. The truth value, denoted $V(C_i)$ (where C_i stands as a label for a cell, $i=1,\dots,47$)

$$V(C_i) \equiv \varphi(s_1,s_2) \equiv (s_1^2+s_2^2 \leq 1) \wedge (s_2 = s_1) \tag{2.18}$$

is evaluated for each one of the cells. The results of this calculation are summarised in the last column of the Table 2.1, where T stands for *true* and F stands for *false*. We see that when $x_1=\alpha_2=-\frac{\sqrt{2}}{2}$, $x_1\in(\alpha_2,\alpha_3)\equiv(-\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2})$ and $x_1=\alpha_3=\frac{\sqrt{2}}{2}$, there exists a cell $C\subset\mathbb{R}^2$ where $V(C)$ evaluates to *true*, i.e. the straight line $(x_2=x_1)$ lies inside the unit disk $(x_1^2+x_2^2\leq 1)$. Hence, as expected, the equivalent quantifier-free expression $\Phi(x_1)$

(see (2.16)) is:

$$\Phi(x_1) \equiv \left(-\frac{\sqrt{2}}{2} \leq x_1 \leq \frac{\sqrt{2}}{2} \right). \tag{2.19}$$

In this intuitive example, it was trivial to perform the projection of the “singularities” of algebraic curves defined by $f_1(x_1,x_2) = 0$ and $f_2(x_1,x_2) = 0$, to find the roots of the polynomials and to evaluate them at rational sample points. General projection operation, polynomial root isolation and evaluation of algebraic sample points are more involved and are discussed in greater detail and rigour in Section 2.3.3.3.

2.3.3.2 Definitions

In this section, we provide various definitions used in the description of the algorithm that constructs cylindrical algebraic decomposition (see Section 2.3.3.3) by, essentially, following Arnon et al. (1984).

Construction of cylindrical algebraic decomposition requires a formal and systematic treatment of systems of equations and inequalities. Hence, a notion of a *semi-algebraic* set is important. A set is *semi-algebraic* if it can be constructed by finitely many applications of union, intersection and complementation operations on sets of the form

$$\{x \in \mathbb{R}^n \mid f(x) \geq 0\}, \tag{2.20}$$

where $f \in \mathbb{R}[x_1, \dots, x_n]$ is an n-variate polynomial.

One of the important properties of semi-algebraic sets is that they are closed under projection, i.e. the projection of a semi-algebraic set to a lower dimensional space is also semi-algebraic. This property is of crucial importance in the projection phase of the cylindrical algebraic decomposition algorithm.

Now, we introduce some definitions that are useful for describing how algebraic surfaces and curves partition the space $\mathbb{R}^n$. These definitions are needed as building blocks for a formal definition of cylindrical algebraic decomposition of $\mathbb{R}^n$.

Let a *region*, R, be a non-empty connected subset of $\mathbb{R}^n$. Then the set:

$$Z(R) = R \times \mathbb{R} = \{(\alpha,x) \mid \alpha \in R, x \in \mathbb{R}\} \tag{2.21}$$

is called a *cylinder* over the region R. Now, let f, f_1, f_2 be continuous and real-valued functions on region R with $f_1(\alpha) < f_2(\alpha)\ \forall \alpha \in R$. Then, an *f-section* of the cylinder $Z(R)$

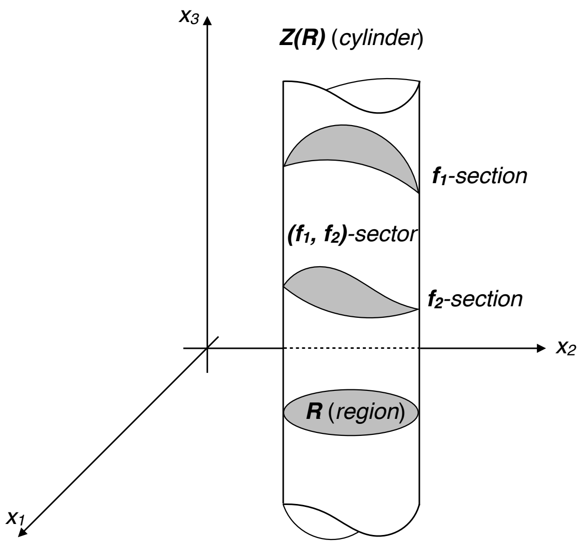


Fig. 2.2 A geometrical interpretation of the definitions of *region*, *cylinder*, *sections*, *sector*, *stack*.

is the set:

$$\{(\alpha, f(\alpha)) \mid \alpha \in R\}, \tag{2.22}$$

which is the graph of f over R. Similarly, an (f_1, f_2)-*sector* of the cylinder $Z(R)$ is the set:

$$\{(\alpha, \beta) \mid \alpha \in R, f_1(\alpha) < \beta < f_2(\alpha)\}. \tag{2.23}$$

Clearly, $f-$sections and (f_1, f_2)-sectors of cylinders are regions. For a geometrical interpretation of these definitions, see Figure 2.2.

Now, let $\mathbb{X} \subseteq \mathbb{R}^n$. Then a *decomposition* of $\mathbb{X}$ is a finite collection of disjoint regions whose union is $\mathbb{X}$:

$$\mathbb{X} = \bigcup_{i=1}^{k} \mathbb{X}_i,\ \mathbb{X}_i \cap \mathbb{X}_j = \emptyset,\ i \neq j. \tag{2.24}$$

Consequently, it is obvious that the set of continuous, real-valued functions defined over the region R, $f_1 < f_2 < \ldots < f_k, \forall x \in R$, naturally induces a decomposition of the cylinder $Z(R)$ consisting of the following regions:

(i) (f_i, f_{i+1})-sectors of $Z(R)$, $0 \leq i \leq k$ with $f_0 = -\infty$ and $f_{k+1} = +\infty$.

(ii) f_i-sections of $Z(R)$, $1 \leq i \leq k$.

Such a decomposition is called a *stack* over R. Also, notice the strict inequalities $f_1 < f_2 < \ldots < f_k, \forall x \in R$ in the definition of the stack. Geometrically, this just means that the graphs of different functions must not intersect each over R (see Figure 2.2).

Finally, we can give a definition of a *cylindrical* decomposition. A decomposition $\mathbb{D}$ of $\mathbb{R}^n$ is *cylindrical* if:

(i) $n = 1$: $\mathbb{D}$ is a partition of a real line $\mathbb{R}^1$ into a finite set of numbers and the finite and infinite open intervals bounded by these numbers.

(ii) $n > 1$: $\mathbb{D}' = \mathbb{F}_1 \cup \ldots \cup \mathbb{F}_m$ is a cylindrical decomposition of $\mathbb{R}^{n-1}$, and over each $\mathbb{F}_i$, there is a stack which is a subset of the decomposition $\mathbb{D}$.

From the definition of cylindrical decomposition, it is obvious that any cylindrical decomposition of $\mathbb{R}^n$ induces a unique cylindrical decomposition of $\mathbb{R}^{n-1}$ all the way down to $\mathbb{R}^1$.

A decomposition is *algebraic* if each of its regions is a semi-algebraic set. When a decomposition is defined by a set of polynomials $f_1, \ldots, f_k \in \mathbb{R}[x_1, \ldots, x_n]$, it is algebraic since all the boundaries in the decomposition are zero sets of those polynomials.

A *cylindrical algebraic decomposition (CAD)* of $\mathbb{R}^n$ is a decomposition that is both cylindrical and algebraic. The components of CAD are called *cells*. For an example of CAD of $\mathbb{R}^2$, see Figure 2.1 (c). This CAD of $\mathbb{R}^2$ induces CAD of $\mathbb{R}^1$ which is represented in Figure 2.1 (b) by black dots and open intervals between them on the real axis.

Now, let $\mathbb{X} \subseteq \mathbb{R}^n$, and $f \in \mathbb{R}[x_1, \ldots, x_n]$ be an n-variate polynomial. Then f is *sign-invariant* on $\mathbb{X}$ if one of the following conditions hold:

(i) $\forall x \in \mathbb{X} : f(x) > 0$ ("f has a positive sign on $\mathbb{X}$")

(ii) $\forall x \in \mathbb{X} : f(x) = 0$ ("f has a zero sign on $\mathbb{X}$")

(iii) $\forall x \in \mathbb{X} : f(x) < 0$ ("f has a negative sign on $\mathbb{X}$")

The set $\mathscr{F} = \{f_1, \ldots, f_r\} \in \mathbb{R}[x_1, \ldots, x_n]$ of polynomials is *invariant* on $\mathbb{X}$ if each $f_i \in \mathscr{F}$ is sign-invariant on $\mathbb{X}$. Also, set $\mathbb{X}$ is *$\mathscr{F}$-invariant* if $\mathscr{F}$ is invariant on $\mathbb{X}$.

Definitions and concepts discussed in this section are of crucial importance in describing the algorithm constructing the cylindrical algebraic decomposition (see Section 2.3.3.3).

2.3.3.3 Cylindrical algebraic decomposition

In this section, we give a description of the algorithm that, given a set of polynomials $\mathscr{F}$, constructs the cylindrical algebraic decomposition (CAD) (Collins, 1975). Discussion here formalises the one for the intuitive example in Section 2.3.3.1. In Section 2.3.3.4, it will be shown how the CAD (obtained by the algorithm discussed in this section) is used for quantifier elimination.

CAD ALGORITHM (Collins, 1975)

Input: $\mathscr{F} = \{f_1, \dots, f_r\} \in \mathbb{R}[x_1, \dots, x_n]$, a set of n-variate polynomials.
Output: $\mathscr{F}$-invariant CAD $\mathbb{D}_n$ of $\mathbb{R}^n$, i.e. all the polynomials in the set $\mathscr{F}$ have the same sign in each of the cells of $\mathbb{D}_n$.
This algorithm consists of three major phases: *projection*, *base* and *extension*.

- **PROJECTION PHASE**

  Let *PROJ* denote the *projection operator*. Then, the algorithm starts by computing the set

$$PROJ(\mathscr{F}) \equiv \mathscr{F}_1 \subset \mathbb{R}[x_1, \dots, x_{n-1}] \tag{2.25}$$

  such that for any $\mathscr{F}_1$-invariant CAD $\mathbb{D}_{n-1}$ of $\mathbb{R}^{n-1}$, there exists an $\mathscr{F}$-invariant CAD $\mathbb{D}_n$ of $\mathbb{R}^n$ which induces $\mathbb{D}_{n-1}$. In general, the algorithm performs projection operation *PROJ* $(n-1)$ times to produce the sets

$$\begin{aligned}
&PROJ(\mathscr{F}) \equiv \mathscr{F}_1 \subset \mathbb{R}[x_1, \dots, x_{n-1}], \\
&PROJ^2(\mathscr{F}) \equiv PROJ(PROJ(\mathscr{F})) \equiv PROJ(\mathscr{F}_1) \equiv \mathscr{F}_2 \subset \mathbb{R}[x_1, \dots, x_{n-2}], \\
&\vdots \\
&PROJ^{n-1}(\mathscr{F}) \equiv \mathscr{F}_{n-1} \subset \mathbb{R}[x_1],
\end{aligned}$$

  where $\mathscr{F}_{n-1}$ is a set of univariate polynomials in x_1. As can be seen, each projection operation reduces the number of variables by one. After the projection phase is completed, the construction of $\mathscr{F}_{n-1}$-invariant CAD $\mathbb{D}_1$ of $\mathbb{R}^1$ is performed (this operation is called the *base phase*). Afterwards, successive extensions of CAD $\mathbb{D}_1$ of $\mathbb{R}^1$ to CAD $\mathbb{D}_2$ of $\mathbb{R}^2$ to ... an $\mathscr{F}$-invariant CAD $\mathbb{D}_n$ of $\mathbb{R}^n$ are performed (this operation is called the *extension phase*).

  The aforementioned projection operator *PROJ* must satisfy the stated desired property, i.e. any $PROJ(\mathscr{F}) \equiv \mathscr{F}_1$-invariant CAD $\mathbb{D}_{n-1}$ of $\mathbb{R}^{n-1}$ is induced by some $\mathscr{F}$-invariant CAD $\mathbb{D}_n$ of $\mathbb{R}^n$. It can be seen that, equivalently, *PROJ* operator has to find regions over which the given set of polynomials, $\mathscr{F}$, has a constant number of real roots. This can be formalised by the notion of *delineability*. To do so, consider $f_i \in \mathscr{F}$, and write it in the form

$$f_i(x_1, \dots, x_{n-1}, x_n) = f_i^{d_i}(x_1, \dots, x_{n-1})x_n^{d_i} + \dots + f_i^0(x_1, \dots, x_{n-1}) \tag{2.26}$$

and let $z = (z_1, \dots, z_{n-1}) \in \mathbb{R}^{n-1}$. Then $f_{i,z}(x_n) = f_i(z_1, \dots, z_{n-1}, x_n)$ is the univariate polynomial obtained by substituting z for the first $n-1$ variables.

Also, introduce the *reductum*, $\hat{f}_i^{k_i}$, of a polynomial

$$\hat{f}_i^{k_i}(x_1, \dots, x_{n-1}, x_n) = f_i^{k_i}(x_1, \dots, x_{n-1})x_n^{k_i} + \dots + f_i^0(x_1, \dots, x_{n-1}), \tag{2.27}$$

where $0 \le k_i \le d_i$. Now, we are ready to state the notion of the set $\mathscr{F}$ being delineable on the subset of $\mathbb{R}^n$.

Definition 1 *(taken from Section 8.6.2 of (Mishra, 1993))*
As before, let $\mathscr{F} = \{f_1, \dots, f_r\} \in \mathbb{R}[x_1, \dots, x_{n-1}][x_n]$ be a set of real polynomials, and let $C \subset \mathbb{R}^{n-1}$ be connected. It is said that $\mathscr{F}$ is delineable on C if it satisfies the following properties:

(i) Condition 1: for each $1 \le i \le r$ (i.e., for every polynomial in the set $\mathscr{F}$), the total number of complex roots of $f_{i,z}$ (counting multiplicity and including real roots) remains invariant as z varies over C.

(ii) Condition 2: for each $1 \le i \le r$ (i.e., for every polynomial in the set $\mathscr{F}$), the number of distinct complex roots (including real roots) of $f_{i,z}$ remains invariant as z varies over C.

(iii) Condition 3: for each $1 \le i < j \le r$ (i.e., for every pair of polynomials in the set $\mathscr{F}$), the total number of complex common roots of $f_{i,z}$ and $f_{j,z}$ (counting multiplicity and including real roots) remains invariant as z varies over C. ■

Moreover, according to the Lemma 8.6.3 in Mishra (1993), if the set $\mathscr{F}$ is delineable over $C \subset \mathbb{R}^{n-1}$, then the number of distinct real roots of $\mathscr{F}$ is invariant over C.

Before describing the projection operator *PROJ*, we have to reference several other definitions and results. Let $f, g \in \mathbb{R}[x_1, \dots, x_{n-1}][x_n]$ be n-variate polynomials with $deg(f) = m$, $deg(g) = n$, $m \ge n$ (where $deg(f)$ denotes the degree of the polynomial f in x_n). For $0 \le k < n$, let $\text{subres}_k(f,g) \in \mathbb{R}[x_1, \dots, x_{n-1}][x_n]$ with $deg(\text{subres}_k(f,g)) \le k$ denote the k^{th} *subresultant* of f and g. Each coefficient of $\text{subres}_k(f,g)$ is the determinant of a certain matrix of f and g coefficients — for more details, see Chapter 7 of Mishra (1993). For $0 \le k < n$, the $k-th$ *principal subresultant coefficient* of f and g w.r.t. x_n, denoted $\text{psc}_k^{x_n}(f,g)$, is the coefficient of x_n^k in $\text{subres}_k(f,g)$. Also, let D_{x_n} denote the partial derivative operator w.r.t. x_n.

Additionally, we reference the following result which will be needed to show that the *PROJ* operator satisfies the stated delineability property.

Proposition 1 *(lemma 7.7.9 in (Mishra, 1993)) For all $0 < i \leq n$, the statement that polynomials f and g have a common factor of degree i is equivalent to:*

$$psc_j(f,g) = 0,\ j = 0,\dots,i-1 \text{ and } psc_i(f,g) \neq 0. \tag{2.28}$$

Finally, we can give the definition of the projection operator *PROJ* which, as required, given the set of polynomials $\mathscr{F} \subset \mathbb{R}[x_1,\dots,x_{n-1}][x_n]$, computes another set of $(n-1)$-variate polynomials $PROJ(\mathscr{F}) \subset \mathbb{R}[x_1,\dots,x_{n-1}]$ characterising the maximal connected $\mathscr{F}$-delineable subsets of $\mathbb{R}^{n-1}$ (i.e., subsets of $\mathbb{R}^{n-1}$ over which polynomials in $\mathscr{F}$ have a constant number of real roots).

The projection operator *PROJ* is defined as follows: given $\mathscr{F}$,

$$PROJ(\mathscr{F}) = PROJ_1(\mathscr{F}) \cup PROJ_2(\mathscr{F}) \cup PROJ_3(\mathscr{F}) \tag{2.29}$$

where

- $PROJ_1 = \{f_i^k(x_1,\dots,x_{n-1}) \mid 1 \leq i \leq r,\ 0 \leq k \leq d_i\}$
  Suppose that the set $PROJ_1(\mathscr{F})$ is invariant over $C \in \mathbb{R}^{n-1}$. The invariance of the set $PROJ_1(\mathscr{F})$ implies that the degree w.r.t. x_n of the polynomials (2.27) is constant over C. This is equivalent to the fact that the number of roots of each polynomial is constant over C, and hence *Condition 1* in the Definition 1 of $\mathscr{F}$ being delineable over C is satisfied.
- $PROJ_2 = \{\mathrm{psc}_l^{x_n}(\hat{f}_i^k(x_1,\dots,x_n), D_{x_n}(\hat{f}_i^k(x_1,\dots,x_n))) \mid 1 \leq i \leq r,\ 0 \leq l < k \leq d_i\}$
  According to Proposition 1, the invariance of the set $PROJ_2(\mathscr{F})$ implies that the greatest common divisor of each polynomial and its derivative $(gcd(f, D_{x_n}f))$ has a constant degree. Also, note that the number of distinct zeros of the polynomial $f \in \mathbb{R}[x_1,\dots,x_{n-1}][x_n]$ is given by $deg(f) - deg(gcd(f, D_{x_n}f))$. Hence, together with the invariance of $PROJ_1(\mathscr{F})$, the number of distinct zeros of each polynomial in $\mathscr{F}$ is constant. Hence, *Condition 2* in the Definition 1 of $\mathscr{F}$ being delineable over C is satisfied.
- $PROJ_3 = \{\mathrm{psc}_m^{x_n}(\hat{f}_i^{k_i}(x_1,\dots,x_n), \hat{f}_j^{k_j}(x_1,\dots,x_n)) \mid 1 \leq i < j \leq r,\ 0 \leq m \leq k_i \leq d_i,\ 0 \leq m \leq k_j \leq d_j\}$
  Again, according to Proposition 1, the invariance of $PROJ_3(\mathscr{F})$, together with

the invariance of $PROJ_1(\mathscr{F})$, implies that the number of common zeros of each pair of polynomials in $\mathscr{F}$ is constant. Hence, *Condition 3* in the Definition 1 of $\mathscr{F}$ being delineable over C is satisfied.

Finally, according to the already stated Lemma 8.6.3 in Mishra (1993), we can conclude that the set $PROJ(\mathscr{F})$ characterises the set over which the number of distinct real zeros of the polynomials in $\mathscr{F}$ is constant, as required.

The projection operator *PROJ* that was described in this section tends to produce sets with a large number of polynomials which consequently increases the execution time of the algorithm constructing CAD. Over the years, various improvements to the original projection operator have been proposed in order to address this issue — for more details, see Hong (1990), McCallum (1988), McCallum (1998), Brown (2001).

- **BASE PHASE**:
Input: output of the projection phase, $PROJ^{n-1}(\mathscr{F}) \equiv \mathscr{F}_{n-1} \subset \mathbb{R}[x_1]$, which is a set of monovariate polynomials.
Output: $\mathscr{F}_{n-1}$-invariant CAD $\mathbb{D}_1$ of $\mathbb{R}^1$.
Zeros of monovariate polynomials in $\mathscr{F}_{n-1}$ define a sign-invariant decomposition of $\mathbb{R}^1$, i.e. CAD $\mathbb{D}_1$ of $\mathbb{R}^1$. In this phase, for all the cells $C_i \in \mathbb{D}_1$, an exact representation of a particular algebraic point (called *sample point* s_i) belonging to that cell is found. To do so, distinct real roots α_i, $i = 1,\dots,k$ of all the polynomial in $\mathscr{F}_{n-1}$ are isolated (via, say, Sturm Sequence based method — for more details, see Section 8.4 in (Mishra, 1993)):

$$-\infty < \alpha_1 < \alpha_2 < \ldots < \alpha_k < +\infty \tag{2.30}$$

with

$$\begin{aligned}
&\alpha_1 \in (r_1, p_1],\\
&\alpha_2 \in (r_2, p_2],\\
&\alpha_3 \in (r_3, p_3],\\
&\vdots\\
&\alpha_k \in (r_k, p_k],
\end{aligned}$$

and $r_1 < p_1 \leq r_2 < p_2 \leq \ldots p_{k-1} \leq r_k < p_k,\ r_i, p_i \in \mathbb{Q}, i = 1,\dots,k$ (i.e., end points of each interval are rational numbers).

A sample point s_i for a cell of $\mathbb{D}_1$ that is an open interval can be chosen to be one of the rational number defining the intervals isolating the roots $\alpha_i,\ i = 1,\dots,k$, i.e. $s_i = r_i$ or $s_i = p_i$ for $i = 1,\dots,k$.

A sample point s_i for a cell of $\mathbb{D}_1$ that is a root $\alpha_i,\ i = 1,\dots,k$ might be represented by a minimal polynomial $M_{\alpha_i}(x)$ and an isolating interval $(a_i,b_i), a_i,b_i \in \mathbb{Q}$ such that the interval (a_i,b_i) contains no other real zeros of $M_{\alpha_i}(x)$ other than α_i.

- **EXTENSION PHASE**:
  Input: a list of sample points s_i describing the CAD $\mathbb{D}_1$ of $\mathbb{R}^1$.
  Output: CAD $\mathbb{D}_n$ of $\mathbb{R}^n$, described by a list of cells (indexed in some way) and their sample points.
  In this step, the base phase is used repetitively to "lift" a sign-invariant decomposition $\mathbb{D}_{i-1}$ of $\mathbb{R}^{i-1}$ to a sign-invariant decomposition $\mathbb{D}_i$ of $\mathbb{R}^i$.

  Initially, consider the lift from $\mathbb{D}_1$ of $\mathbb{R}^1$ to $\mathbb{D}_2$ of $\mathbb{R}^2$. Let C be a cell of $\mathbb{D}_1$ with a sample point $x_1 = s$. Then the polynomials in the set $\mathscr{F}_{n-2} \equiv PROJ^{n-2}(\mathscr{F})$ are evaluated at the sample point $x_1 = s$. This, in turn, results in a set of monovariate polynomials in x_2. Then, the base phase is performed on those monovariate polynomials (i.e., root isolation and sample point creation). Hence, second components of the sample points of the CAD $\mathbb{D}_2$ of $\mathbb{R}^2$ are constructed.

  Consequently, the extension process is then repeated to construct sample points of the cells of the decomposition $\mathbb{D}_3$ of $\mathbb{R}^3$, ..., $\mathbb{D}_n$ of $\mathbb{R}^n$.

  After the extension phase is completed, what we end up is a list of cells and their sample points, i.e. CAD $\mathbb{D}_n$ of $\mathbb{R}^n$. ■.

2.3.3.4 Quantifier elimination algorithm based on cylindrical algebraic decomposition

Recall the first order formula (2.7)

$$Q_1 x_1 \dots Q_n x_n \varphi(p_1,\dots,p_m,x_1,\dots,x_n), \tag{2.31}$$

where $Q_i \in \{\forall,\exists\}$ are either universal or existential quantifiers and φ is a quantifier-free formula constructed by conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) of atomic formulas of the form $f_j\,\rho\,0,\ j = 1,\dots,r$ (where $f_j \in \mathbb{R}[p_1,\dots,p_m,x_1,\dots,x_n]$ is a polynomial and $\rho \in \{=,\neq,<,\leq\}$ is a relational operator). Also, let $\mathscr{F} = \{f_1,\dots,f_r\} \in \mathbb{R}[p_1,\dots,p_m,x_1,\dots,x_n]$ be the set of polynomials that appear in φ.

Then Collins' quantifier elimination algorithm, based on cylindrical algebraic decomposition (Collins, 1975), eliminates quantifiers $Q_1,\dots,Q_n$ from the expression (2.31) as follows:

QE by CAD algorithm (Collins, 1975)

Input: a first order formula (2.31).
Output: a quantifier-free formula $\Phi(p_1,\dots,p_m)$ that is equivalent to (2.31).

- **Step 1**: *CAD construction.* A cylindrical algebraic decomposition $\mathbb{D}_{n+m}$ for $\mathbb{R}^{n+m}$ induced by the set of polynomials $\mathscr{F}$ is built, as described in Section 2.3.3.3.

- **Step 2**: *Evaluation.* Now, let C be a cell of $\mathbb{D}_{n+m}$, and let $s=(s_1,\dots,s_m,s_{m+1},\dots,s_{m+n})$ be a sample point of C (where $s_i, i=1,\dots,m+n$ is the $i-$th coordinate of the sample point). Denote $V(C)$ the truth value (*true* or *false*) of $\varphi(s_1,\dots,s_m,s_{m+1},\dots,s_{m+n})$. Because of the way the CAD is constructed, the value of $V(C)$ is constant throughout the cell C. Hence, in this step, the truth values of all cells C in $\mathbb{D}_{n+m}$ are evaluated.

- **Step 3**: *Propagation.* For $k=n-1,n-2,\dots,0$, determine the truth values of the cells of the CAD $\mathbb{D}_{k+m}$ from the truth values of the cells of the CAD $\mathbb{D}_{k+m+1}$ by using the following procedure:

  Let C be a cell of $\mathbb{D}_{k+m}, 0\le k\le n-1$, and let $C_1,\dots,C_l$ be the cells in the stack over C. Then:

  – If $Q_{k+1}\equiv\exists$, then $V(C)=\bigvee_{i=1}^{l} V(C_i)$.

  – If $Q_{k+1}\equiv\forall$, then $V(C)=\bigwedge_{i=1}^{l} V(C_i)$.

  Essentially, in this step, the truth values of the cells of $\mathbb{D}_{k+m+1}$ are propagated to the truth values of the cells of $\mathbb{D}_{k+m}$.

- **Step 4**: *Solution.* Finally, a quantifier-free expression $\Phi(p_1,\dots,p_m)$ can be obtained. Let C be a cell of CAD $\mathbb{D}_m$, and define:

$$S=\{C\in\mathbb{D}_m \mid V(C)=true\}. \tag{2.32}$$

  Then:

$$\boxed{\Phi(p_1,\dots,p_m)\equiv\bigcup_{C\in S} C. \blacksquare} \tag{2.33}$$

As can be seen from the description of the algorithm, full CAD of $\mathbb{R}^{n+m}$ is constructed first (**step** 1) and truth evaluations are performed later (**steps** $2,3$). Construction of CAD, in general, is a computationally expensive procedure. Hence, the execution time of this quantifier elimination algorithm can be improved by combining CAD construction with truth evaluation so that parts of the CAD are constructed only as needed to perform truth evaluation and CAD construction is aborted as soon as no more truth evaluation is needed. This is achieved by making use of more information contained in the input formula (2.31), including its quantification structure, Boolean connectives and the absence of some variables from some of the polynomials. This is called *partial CAD for quantifier elimination* — for more details, see Collins and Hong (1991).

We have covered all the technical details needed to provide a quantifier elimination by cylindrical algebraic decomposition example that illustrates all the mathematical machinery discussed in previous sections.

2.3.3.5 Full example

Let

$$
\begin{aligned}
f_1(x_1,x_2) &= x_2^2 - 2x_1x_2 + x_1^4, \\
f_2(x_1,x_2) &= (2431x_1 - 3301)x_2 - 2431x_1 + 2685.
\end{aligned}
$$

Now, consider the quantifier elimination problem

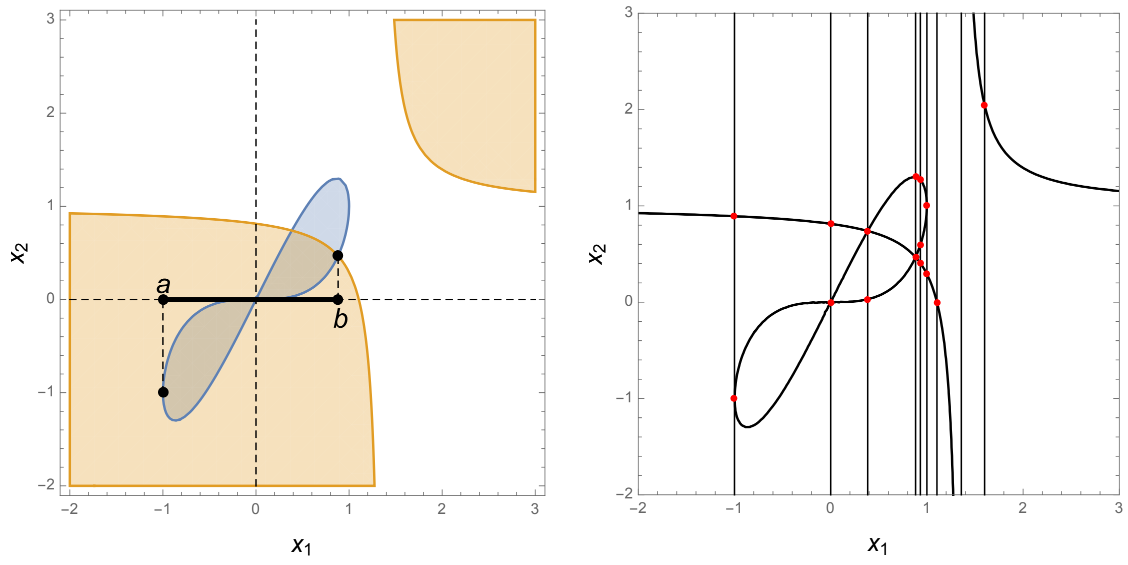
$$
\Phi(x_1) \equiv \exists x_2 \varphi(x_1,x_2) \equiv \exists x_2 : (f_1(x_1,x_2) \leq 0) \wedge (f_2(x_1,x_2) \geq 0) \tag{2.34}
$$

which asks us to find for which values of the variable x_1 there exists a value of the variable x_2 such that the polynomial $f_1(x_1,x_2)$ is non-positive and the polynomial $f_2(x_1,x_2)$ is non-negative. Graphical solution to this problem is illustrated in Figure 2.3 (a).

We will arrive at the solution of this problem by using the full quantifier elimination by cylindrical algebraic decomposition algorithm, as described in Sections 2.3.3.3 and 2.3.3.4.

- **Step 1**: *CAD construction*
  Firstly, CAD $\mathbb{D}_2$ of $\mathbb{R}^2$ is built, which, as described in Section 2.3.3.3, consists of the *projection*, *base* and *extension* phases.

  Projection phase



(a) Graphical solution to the quantifier elimination problem (2.34).

(b) CAD $\mathbb{D}_2$ of $\mathbb{R}^2$.

Fig. 2.3 In figure (a), the blue region represents $f_1(x_1,x_2) \leq 0$ and the orange one corresponds to $f_2(x_1,x_2) \geq 0$. Hence, the solution to the quantifier elimination problem (2.34) is given by the interval $a \leq x_1 \leq b$. Figure (b) illustrates the cylindrical algebraic decomposition of $\mathbb{R}^2$ induced by the polynomials $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$.

Let $\mathscr{F} = \{f_1, f_2\} \in \mathbb{R}[x_1,x_2]$. Reductum of the polynomials

$$\begin{aligned} f_1(x_1,x_2) &= f_1^2(x_1)x_2^2 + f_1^1(x_1)x_2 + f_1^0(x_1), \\ f_2(x_1,x_2) &= f_2^1(x_1)x_2 + f_2^0(x_1), \end{aligned}$$

(in alignment with (2.26)), as described by (2.27), are

$$\hat{f}_1^0 = x_1^4,\ \hat{f}_1^1 = -2x_1x_2 + x_1^4,\ \hat{f}_1^2 = f_1 = x_2^2 - 2x_1x_2 + x_1^4,$$

and

$$\hat{f}_2^0 = -2431x_1 + 2685,\ \hat{f}_2^1 = f_2 = (2431x_1 - 3301)x_2 - 2431x_1 + 2685,$$

respectively. Then:

$$PROJ_1(\mathscr{F}) \equiv \{f_i^k(x_1) \mid 1 \le i \le 2,\ 0 \le k \le \deg(f_i)\}$$
$$= \{1, -2x_1, x_1^4, (2431x_1 - 3301), (-2431x_1 + 2685)\}$$
$$PROJ_2(f_1) \equiv \{\mathrm{psc}_l^{x_2}(\hat{f}_1^k(x_1,x_2), D_{x_2}(\hat{f}_1^k(x_1,x_2))) \mid 0 \le l < k \le 2\}$$

$\underline{l = 0, k = 1}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^1(x_1,x_2), D_{x_2}(\hat{f}_1^1(x_1,x_2))) = \mathrm{psc}_0^{x_2}(-2x_1x_2 + x_1^4, -2x_1) = -2x_1$$

$\underline{l = 0, k = 2}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^2(x_1,x_2), D_{x_2}(\hat{f}_1^2(x_1,x_2))) = \mathrm{psc}_0^{x_2}(x_2^2 - 2x_1x_2 + x_1^4, 2x_2 - 2x_1)$$
$$= 4x_1^2(x_1 - 1)(x_1 + 1)$$

$\underline{l = 1, k = 2}$

$$\mathrm{psc}_1^{x_2}(\hat{f}_1^2(x_1,x_2), D_{x_2}(\hat{f}_1^2(x_1,x_2))) = \mathrm{psc}_1^{x_2}(x_2^2 - 2x_1x_2 + x_1^4, 2x_2 - 2x_1) = 2$$

$$PROJ_2(f_2) \equiv \{\mathrm{psc}_l^{x_2}(\hat{f}_2^k(x_1,x_2), D_{x_2}(\hat{f}_2^k(x_1,x_2))) \mid 0 \le l < k \le 1\}$$

$\underline{l = 0, k = 1}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_2^1(x_1,x_2), D_{x_2}(\hat{f}_2^1(x_1,x_2)))$$
$$= \mathrm{psc}_0^{x_2}((2431x_1 - 3301)x_2 - 2431x_1 + 2685, 2431x_1 - 3301)$$
$$= 2431x_1 - 3301$$

$$PROJ_3(\mathscr{F}) \equiv \{\mathrm{psc}_m^{x_2}(\hat{f}_1^{k_1}(x_1,x_2), \hat{f}_2^{k_2}(x_1,x_2)) \mid 0 \le m \le k_1 \le \deg(f_1) = 2,\ 0 \le m \le k_2 \le \deg(f_2) = 1\}$$

$\underline{m = 0, k_1 = 0, k_2 = 0}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^0(x_1,x_2), \hat{f}_2^0(x_1,x_2)) = \mathrm{psc}_0^{x_2}(x_1^4, -2431x_1 + 2685) = -2431x_1 + 2685$$

$\underline{m = 0, k_1 = 0, k_2 = 1}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^0(x_1,x_2), \hat{f}_2^1(x_1,x_2)) = \mathrm{psc}_0^{x_2}(x_1^4, (2431x_1 - 3301)x_2 - 2431x_1 + 2685) =$$

{undefined, since $\hat{f}_2^1(x_1,x_2)$ has a higher degree in x_2 than $\hat{f}_1^0(x_1,x_2)$}

$\underline{m = 0, k_1 = 1, k_2 = 0}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^1(x_1,x_2), \hat{f}_2^0(x_1,x_2)) = \mathrm{psc}_0^{x_2}(-2x_1x_2 + x_1^4, -2431x_1 + 2685)$$
$$= -2431x_1 + 2685$$

$\underline{m = 0, k_1 = 1, k_2 = 1}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^1(x_1,x_2), \hat{f}_2^1(x_1,x_2))$$
$$= \mathrm{psc}_0^{x_2}(-2x_1x_2 + x_1^4, (2431x_1 - 3301)x_2 - 2431x_1 + 2685)$$
$$= -x_1\left(2431x_1^4 - 3301x_1^3 - 4862x_1 + 5370\right)$$

$\underline{m = 0, k_1 = 2, k_2 = 0}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^2(x_1,x_2), \hat{f}_2^0(x_1,x_2)) = \mathrm{psc}_0^{x_2}(x_2^2 - 2x_1x_2 + x_1^4, -2431x_1 + 2685)$$
$$= -2431x_1 + 2685$$

$\underline{m = 0, k_1 = 2, k_2 = 1}$

$$\mathrm{psc}_0^{x_2}(\hat{f}_1^2(x_1,x_2), \hat{f}_2^1(x_1,x_2))$$
$$= \mathrm{psc}_0^{x_2}(x_2^2 - 2x_1x_2 + x_1^4, (2431x_1 - 3301)x_2 - 2431x_1 + 2685)$$
$$= (13x_1 - 5)(17x_1 - 15)\left(26741x_1^4 - 38742x_1^3 - 8854x_1^2 - 51552x_1 + 96123\right)$$

$\underline{m = 1, k_1 = 1, k_2 = 1}$

$$\mathrm{psc}_1^{x_2}(\hat{f}_1^1(x_1,x_2), \hat{f}_2^1(x_1,x_2))$$
$$= \mathrm{psc}_1^{x_2}(-2x_1x_2 + x_1^4, (2431x_1 - 3301)x_2 - 2431x_1 + 2685)$$
$$= 2431x_1 - 3301$$

$\underline{m = 1, k_1 = 2, k_2 = 1}$

$$\mathrm{psc}_1^{x_2}(\hat{f}_1^2(x_1,x_2), \hat{f}_2^1(x_1,x_2))$$
$$= \mathrm{psc}_1^{x_2}(x_2^2 - 2x_1x_2 + x_1^4, (2431x_1 - 3301)x_2 - 2431x_1 + 2685)$$
$$= 2431x_1 - 3301$$

Cell index	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
x_1	$(-\infty,-1)$	-1	$(-1,0)$	0	$\left(0,\frac{5}{13}\right)$	$\frac{5}{13}$	$\left(\frac{5}{13},\frac{15}{17}\right)$	$\frac{15}{17}$	$\left(\frac{15}{17},r_1\right)$	r_1
s_1	-2	-1	$-\frac{1}{2}$	0	$\frac{1}{4}$	$\frac{5}{13}$	$\frac{1}{2}$	$\frac{15}{17}$	$\frac{9}{10}$	r_1
Cell index	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	
x_1	$(r_1,1)$	1	$\left(1,\frac{2685}{2431}\right)$	$\frac{2685}{2431}$	$\left(\frac{2685}{2431},\frac{3301}{2431}\right)$	$\frac{3301}{2431}$	$\left(\frac{3301}{2431},r_2\right)$	r_2	$(r_2,+\infty)$	
s_1	$\frac{19}{25}$	1	$\frac{21}{20}$	$\frac{2685}{2431}$	$\frac{6}{5}$	$\frac{3301}{2431}$	$\frac{3}{2}$	r_2	2	

Table 2.2 CAD $\mathbb{D}_1$ of $\mathbb{R}^1$.

Hence

$$\begin{aligned}
PROJ(\mathscr{F}) =& PROJ_1(\mathscr{F}) \cup PROJ_2(\mathscr{F}) \cup PROJ_3(\mathscr{F})\\
=&\{1, -2x_1, x_1^4, (2431x_1-3301), (-2431x_1+2685), 4x_1^2(x_1-1)(x_1+1), 2,\\
&-x_1(2431x_1^4-3301x_1^3-4862x_1+5370),\\
&(13x_1-5)(17x_1-15)(26741x_1^4-38742x_1^3-8854x_1^2-51552x_1+96123)\}\\
\equiv&\mathscr{F}_1 \subset \mathbb{R}[x_1],
\end{aligned}$$

a set of univariate polynomials in x_1.

Base phase

In this phase, the $PROJ(\mathscr{F}) \equiv \mathscr{F}_1$-invariant CAD $\mathbb{D}_1$ of $\mathbb{R}^1$ is constructed, i.e. the real line $\mathbb{R}$ is split into points and open intervals where each of the polynomials in $\mathscr{F}_1$ have a constant sign. This decomposition is obviously defined by the real roots of the polynomials in $\mathscr{F}_1$

$$-1,\ 0,\ \frac{5}{13},\ \frac{15}{17},\ r_1,\ 1,\ \frac{2685}{2431},\ \frac{3301}{2431},\ r_2, \tag{2.35}$$

where r_1 $(0.9 < r_1 < 0.95)$ and r_2 $(1.5 < r_2 < 1.6)$ are real roots of the polynomial

$$2431x_1^4 - 3301x_1^3 - 4862x_1 + 5370 \in \mathscr{F}_1$$

which can be isolated via Sturm Sequence based method (see Section 8.4 in (Mishra, 1993) for more details). Similarly, it can be found that the polynomial

$$26741x_1^4 - 38742x_1^3 - 8854x_1^2 - 51552x_1 + 96123 \in \mathscr{F}_1$$

has no real roots. These real roots (2.35) define an $\mathscr{F}_1$-invariant decomposition of $\mathbb{R}$, consisting of 9 points and 10 open intervals with sample points s_1 given in Table 2.2.

Cell index C_{ij}	C_{71}	C_{72}	C_{73}	C_{74}	C_{75}	C_{76}	C_{77}
Sample point (s_1,s_2)	$\left(\frac{1}{2},0\right)$	$\left(\frac{1}{2},\frac{1}{2}-\frac{1}{4}\sqrt{3}\right)$	$\left(\frac{1}{2},\frac{1}{2}\right)$	$\left(\frac{1}{2},\frac{2939}{4171}\right)$	$\left(\frac{1}{2},\frac{3}{4}\right)$	$\left(\frac{1}{2},\frac{1}{2}+\frac{1}{4}\sqrt{3}\right)$	$\left(\frac{1}{2},1\right)$
$V(C_{ij}) \equiv \varphi(s_1,s_2)$	F	T	T	T	F	F	F

Table 2.3 Quantifier elimination for the stack over the cell $C_7 \in \mathbb{D}_1$.

Extension phase

In this step, the base phase is used repeatedly to "lift" the CAD $\mathbb{D}_1$ of $\mathbb{R}^1$ (obtained in the previous step) to the CAD $\mathbb{D}_2$ of $\mathbb{R}^2$. To illustrate this phase, we will only perform calculations for the cell $C_7 = \left(\frac{5}{13},\frac{15}{17}\right) \in \mathbb{D}_1$ with a sample point $s_1 = \frac{1}{2}$ (calculations for the remaining cells of $\mathbb{D}_1$ are analogous). Firstly, the polynomials in the set $\mathscr{F} = \{f_1(x_1,x_2), f_2(x_1,x_2)\}$ are calculated at the sample point $x_1 = s_1 = \frac{1}{2}$

$$f_1(s_1,x_2) = x_2^2 - x_2 + \frac{1}{16}, \left(\text{real roots } \frac{1}{2} \pm \frac{1}{4}\sqrt{3}\right),$$
$$f_2(s_1,x_2) = -\frac{4171}{2}x_2 + \frac{2939}{2}, \left(\text{real root } \frac{2939}{4171}\right),$$

which results in a pair of univariate polynomials in x_2. Now the base phase (real root isolation and sample point creation) is performed on these polynomials which results in the sample points s_2 for the stack over the cell $C_7 \in \mathbb{D}_1$ (see Table 2.3). After performing this extension procedure for all the cells $C_i \in \mathbb{D}_1, i = 1,\ldots,19$, we obtain an $\mathscr{F}$-invariant CAD $\mathbb{D}_2$ of $\mathbb{R}^2$, as depicted in Figure 2.3 (b).

In the following steps, we again limit our calculations to the stack over the cell $C_7 \in \mathbb{D}_1$ (since calculations for the remaining cells are analogous).

- **Step 2**: *Evaluation*
  In this step, the truth value

$$V(C_{ij}) \equiv \varphi(s_1,s_2) \equiv (f_1(s_1,s_2) \leq 0) \wedge (f_2(s_1,s_2) \geq 0) \tag{2.36}$$

is calculated for all the cells $C_{7j} \in \mathbb{D}_2, j = 1,\ldots,7$ (i.e. for all the cells in the stack over the cell $C_7 \in \mathbb{D}_1$). The results are summarized in Table 2.3, where T stands for *true* and F stands for *false*.

- **Step 3**: *Propagation*
  In this step, the truth values of the cells of the CAD $\mathbb{D}_1$ are determined from the truth values of the cells of the CAD $\mathbb{D}_2$.

Again, consider the cell $C_7 \in \mathbb{D}_1$ with $C_{7j} \in \mathbb{D}_2, j = 1,\dots,7$ being the cells in the stack over C_7. Since the quantifier elimination problem (2.34) has a single existential quantifier

$$V(C_7) = \bigvee_{j=1}^{7} V(C_{7j}) = F \vee T \vee T \vee T \vee F \vee F \vee F = T, \tag{2.37}$$

which implies that when $x_1 \in C_7 = \left(\frac{5}{13}, \frac{15}{17}\right)$, (2.34) evaluates to *true*.

- **Step 4**: *Solution*
  By performing the extension, evaluation and propagation calculations for all the cells $C_i, i = 1,\dots,19$ in the CAD $\mathbb{D}_1$ (which are analogous to the one done for C_7), we eventually arrive at the quantifier-free expression to the quantifier elimination problem (2.34):
  $$\Phi(x_1) \equiv \bigcup_{\{C_i \in \mathbb{D}_1 | V(C_i) = true\}} C_i \equiv -1 \leq x_1 \leq \frac{15}{17}. \tag{2.38}$$

2.4 *MetiTarski*: theorem prover for real-valued functions

2.4.1 Introduction

MetiTarski (Paulson, 2012) is an automatic theorem prover that is designed to prove universally quantified mathematical inequalities containing real-valued functions such as $\log_{10}(x)$, $\exp(x)$, $\sin(x)$, $\cos(x)$, $\tan(x)$, $\arcsin(x)$, $\arccos(x)$, $\arctan(x)$, $\sinh(x)$, $\cosh(x)$, $\tanh(x)$, $\sqrt[n]{x}$. It is based on a combination of *Metis* (Hurd, 2003, 2007), a resolution theorem prover, and a decision procedure for the theory of real closed fields (i.e., a quantifier elimination algorithm). Cursory explanation of how *MetiTarski* produces a proof is as follows:

- **Step 1**: Real-valued functions are eliminated by substituting appropriate upper or lower bounds obtained from Taylor series or continued fraction expansions. In general, a typical proof uses a combination of bounds in order to get a good approximation of the real-valued function over an appropriate range of variables. For example, upper and lower bounds for $\log(x)$ that are valid for $x > 0$ and based on a continued fraction expansion are:
  $$\frac{(47x^3 + 239x^2 + 131x + 3)(x-1)}{12x(x^3 + 12x^2 + 18x + 4)} \leq \log(x) \leq \frac{(3x^3 + 131x^2 + 239x + 47)(x-1)}{12(4x^3 + 18x^2 + 12x + 1)}. \tag{2.39}$$
  Hence, this step converts the problem of proving a universally quantified real-valued

function inequality into the problem of the form

$$\forall x_1 \dots \forall x_n \varphi(x_1, \dots, x_n), \tag{2.40}$$

where (as in (2.7)) φ is a quantifier-free formula constructed by conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) of atomic formulas of the form $f \rho 0$ (where $f \in \mathbb{R}[x_1, \dots, x_n]$ is a polynomial and $\rho \in \{=, \neq, <, \leq\}$ is a relational operator).

- **Step 2**: To prove the problem (2.40), *MetiTarski* calls an external quantifier elimination algorithm implemented in *Z3* (de Moura and Bjørner, 2008), *QEPCAD* (Brown, 2003) or *Mathematica* (Wolfram Research Inc., 2016).

While any system of polynomial equalities and inequalities of the form (2.7) can be proven in principle (if given enough time and memory) by using the quantifier elimination by cylindrical algebraic decomposition algorithm (see Section 2.3.3), the problem of proving a general real-valued function inequality is *undecidable* (Paulson, 2012), i.e. there does not exist an algorithm which could prove all *true* statements of this kind. Hence, we cannot conclude anything from the failure of *MetiTarski* to prove an inequality. Despite this fact, in many situations, it is remarkably useful, as illustrated by the following example.

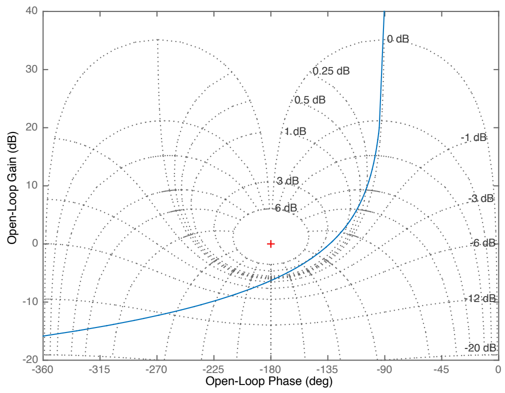
2.4.2 Example: time delay with lag and integrator

In this section, we provide a simple example that illustrates *MetiTarski* capability to prove inequalities involving real-valued functions like $\sin(x)$ and $\cos(x)$. Consider a transfer function of the form:

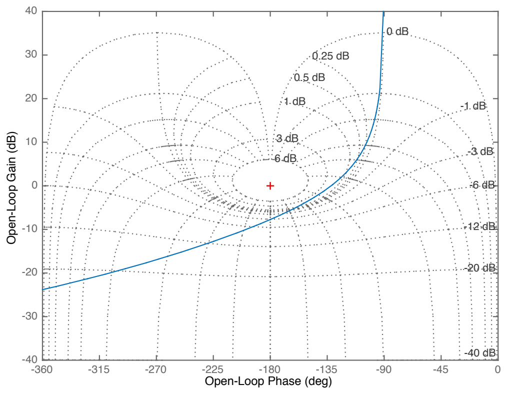
$$L(s) = \frac{e^{-sT_1}}{s(1+sT_2)}. \tag{2.41}$$

Suppose that we are interested in the following verification criterion: what is the largest value r such that, $\forall\, T_1 + T_2 \leq r,\ T_1 \geq 0,\ T_2 \geq 0$, the magnitude of the complementary sensitivity function satisfies:

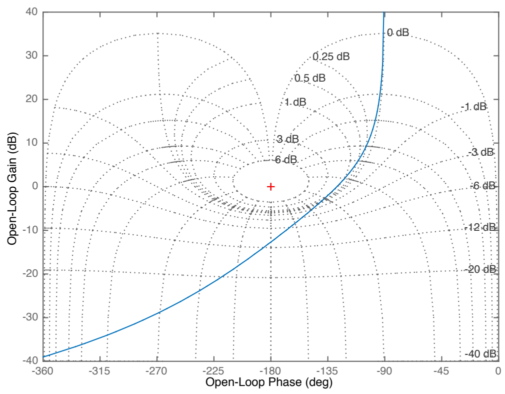
$$|T(j\omega)| = \left| \frac{L(j\omega)}{1+L(j\omega)} \right| \leq \sqrt{2} = 3\text{dB}. \tag{2.42}$$



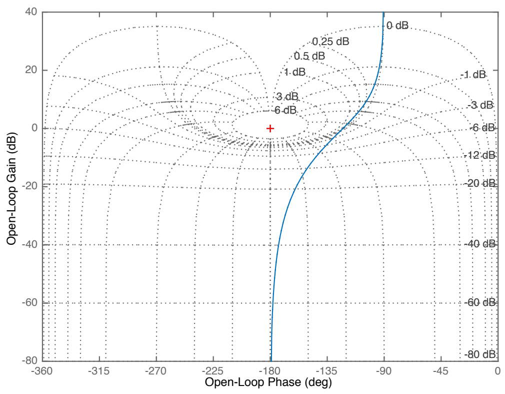
(a) Nichols plot with $T_1 = R,\ T_2 = 0$.



(b) Nichols plot with $T_1 = 0.5,\ T_2 = R - 0.5$.



(c) Nichols plot with $T_1 = 0.25,\ T_2 = R - 0.25$.



(d) Nichols plot with $T_1 = 0,\ T_2 = R$.

Fig. 2.4 Nichols plots for various T_1 and T_2 values satisfying $T_1 + T_2 = R$.

This verification criterion gets translated to the following real-valued function problem that *MetiTarski* is asked to prove:

$$\begin{aligned}\forall x, T_1, T_2 : (T_1 \geq 0) \wedge (T_2 \geq 0) \wedge (T_1 + T_2 \leq r)& \\ \implies (T_2^2 x^2 + 1) \cdot (2x^2 - 4x\sin(T_1 x) + 2T_2^2 x^4 - 4T_2 x^2 \cos(T_1 x) + 1) > 0).&\end{aligned} \tag{2.43}$$

Initially, we ask *MetiTarski* to prove (2.43) for $r = 1$ — it returns *false*. Consequently, we ask *MetiTarski* to prove (2.43) for the following values of $r \in [0,1]$

$$0.5\ (\textit{MetiTarski} \text{ returns } \textit{true}), 0.75\ (\textit{MetiTarski} \text{ returns } \textit{true}),$$
$$0.875\ (\textit{MetiTarski} \text{ returns } \textit{false}), 0.8125\ (\textit{MetiTarski} \text{ returns } \textit{false}),$$
$$0.78125\ (\textit{MetiTarski} \text{ returns } \textit{false}), 0.765625\ (\textit{MetiTarski} \text{ returns } \textit{false}),$$
$$0.7578125\ (\textit{MetiTarski} \text{ returns } \textit{true}), 0.76171875\ (\textit{MetiTarski} \text{ returns } \textit{true}),$$

i.e., we essentially perform a simple bisection search for r. Hence, we find that the largest value of r for which (2.43) holds lies in the interval $0.76171875 \leq r \leq 0.765625$.

Let $R = 0.76171875$. Nichols plots with loci of constant $20\log_{10}|T(s)|$ and $\arg(T(s))$ overlaid are depicted in Figure 2.4 for several cases of T_1 and T_2 values satisfying $T_1 + T_2 = R$. The limiting case is achieved when $T_1 = R$ and $T_2 = 0$ — see Figure 2.4 (a).

Chapter 3

Application of Formal Methods to Control Analysis Problems

In this chapter, robustness criteria of interest are defined and then translated to the form that automated theorem provers (*Why3*, *MetiTarski*) or one of the quantifier elimination algorithms (Weispfenning's virtual term substitution, quantifier elimination based on cylindrical algebraic decomposition) are capable of verifying. Consequently, this allows us to analyse robustness criteria from a different perspective and, in some cases, obtain results that standard control analysis techniques are not capable of.

In Section 3.1, it is shown how the *Why3* verification framework can be used to prove Lyapunov stability of an autonomous system implemented graphically in *Simulink*. Section 3.2 shows how avoidance of various exclusion regions by the open loop transfer function can be expressed as a simple quantifier elimination problem. In Section 3.3, quantifier elimination algorithms are used for analysing stability of uncertain systems via calculation of the structured singular value μ. Additionally, contrast is drawn between the proposed quantifier elimination based method and the standard methods (like branch and bound algorithms) for computing the value of μ. Then, in Section 3.4, we show how a general optimisation problem can be converted to a quantifier elimination one. This is then used for obtaining an explicit MPC solution which consequently allows verification of properties like recursive feasibility. Moreover, in Section 3.5, it is shown how a Linear Temporal Logic (LTL) specification can be converted to a quantifier elimination problem. This quantifier elimination problem is then used to find feasible parameter sets in cases when standard Linear Temporal Logic (LTL) approaches fail, like nonlinearly parametrized systems or LTL specifications that result in non-convex feasible parameter sets.

In Section 3.6.1, a backup flight control scheme developed for the RECONFIGURE project is described. Initially, in Section 3.6.2, a quantifier-elimination-based verification framework is applied to this control law in order to verify certain properties of the system at particular points in the flight envelope. In Section 3.6.3, a verification framework based on quantifier elimination is developed to clear various criteria of interest throughout the whole flight envelope for the same backup flight control law. Consequently, this shows that, in some particular instances, verification frameworks based on formal methods are capable of dealing with real world industrial problems.

3.1 Verification of properties of control systems implemented graphically with *Why3*-based verification framework

3.1.1 Introduction

The capability to automatically clear verification criteria of control systems implemented graphically in *Simulink* is undoubtedly appealing from a practical perspective. This motivates the development of a verification framework that would take a system implemented in *Simulink* (together with additional blocks in the *Simulink* diagram to represent verification criteria of interest) as an input and then would automatically create a *Why3* theory (see Section 2.2) representing this graphical implementation. Finally, the *Why3* tool could then be used to check whether or not the criteria are met by passing the corresponding *Why3* theory to SMT provers it interfaces with. In particular, this kind verification scheme would be a powerful tool to clear verification requirements for the backup flight control law developed in Maciejowski et al. (2016) for the RECONFIGURE project (Goupil et al., 2014).

Therefore, in Section 3.1.2, we develop a verification framework of this nature while simultaneously carrying out preliminary tests of its capability by using it to check asymptotic stability of several instances of autonomous linear discrete time systems. Unfortunately, even for these simple systems, the *Why3* verification framework often runs into computational issues (mainly running out of time) at the SMT prover stage. The reason for this is discussed in Section 3.1.3. Fortunately, also Section 3.1.3, we show that all the verification criteria that the *Why3* verification framework failed to clear was cleared with no issues by using an approach based on quantifier elimination algorithms discussed in Section 2.3.3.

Hence, verification approach that utilises quantifier elimination methods is taken advantage of throughout the rest of this chapter. Moreover, as will be shown later in Sections 3.6.2 and 3.6.3, this quantifier-elimination-based approach (rather than the *Why3* verification framework) is used to successfully clear verification criteria of interest for the backup flight control law developed in Maciejowski et al. (2016).

3.1.2 *Why3* verification framework and its application to verify asymptotic stability

Consider the autonomous linear discrete time system

$$x(k+1) = Ax(k),\ k = 0, 1, 2,\ \ldots \tag{3.1}$$

and the candidate Lyapunov function of the form

$$V(x(k)) = x(k)^T P x(k), \tag{3.2}$$

where P is chosen to be a positive definite matrix. In order to show that the origin is a globally asymptotically stable equilibrium point for the system (3.1) with a particular matrix A, it is sufficient to show that the Lyapunov function (3.2) satisfies the following requirements:

1. *Req. 1:* $\forall x(k) \neq 0,\ V(x(k+1)) - V(x(k)) < 0.$
2. *Req. 2:* $\forall x(k) \neq 0,\ V(x(k)) > 0.$
3. *Req. 3:* $V(x(k)) = 0$ if $x(k) = 0.$

Obviously, requirements *Req. 2* and *Req. 3* are already satisfied (since the matrix P is chosen to be positive definite and the Lyapunov function is chosen as in (3.2)). Regardless, we will still verify these requirements with the *Why3* verification framework, together with the requirement *Req. 1* for specific examples of systems of the form (3.1). The high-level overview of the *Why3* verification framework (implemented by us analogously to the approach considered in Araiza-Illan et al. (2014)) is illustrated in Figure 3.1. In this framework, system (3.1) is firstly implemented graphically as a *Simulink* model. Then high level verification requirements (such as *Req. 1, 2, 3*) are expressed by adding custom *'Require'* type blocks to this *Simulink* diagram. Finally, this annotated diagram is automatically translated to a *Why3* theory with verification goals which, if discharged by one of the SMT provers, verifies that the system meets the required criteria. In the following sections, we discuss the steps of this framework in more detail.

Step 1

First of all, the system of interest has to be represented graphically as a *Simulink* model. For

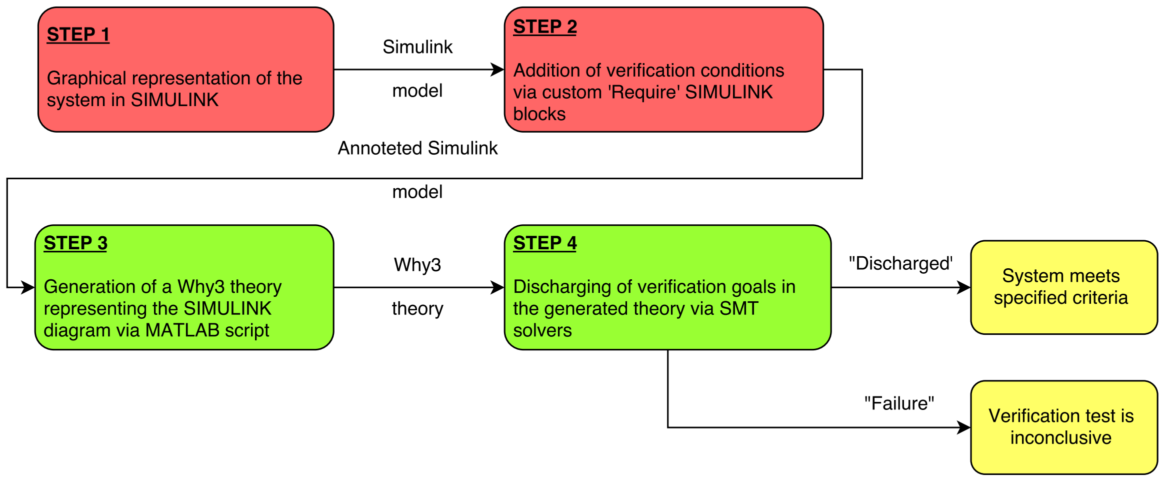

STEP 1
Graphical representation of the system in SIMULINK
Simulink model
STEP 2
Addition of verification conditions via custom 'Require' SIMULINK blocks
Annoteted Simulink model
STEP 3
Generation of a Why3 theory representing the SIMULINK diagram via MATLAB script
Why3 theory
STEP 4
Discharging of verification goals in the generated theory via SMT solvers
"Discharged'
System meets specified criteria
"Failure"
Verification test is inconclusive

Fig. 3.1 Verification framework for systems expressed graphically in *Simulink*. Red blocks represent actions required by a human user, green blocks indicate automated steps and yellow blocks show the outcome of verification.

simplicity, we assume that all blocks and signals are scalar. Note that this does not impose the constraint on $x(k)$ to be scalar — this will be illustrated in Section 3.1.2.2 for $x(k) \in \mathbb{R}^2$. The following scalar *Simulink* blocks are currently allowed (since we have developed *Why3* theories for these *Simulink* blocks and added the appropriate code to the *MATLAB* translation script used in *step 3*): *'Sum'*, *'Product'*, *'Gain'*, *'Unit delay'*, *'Unit advance'*, *'Saturation'*, various compare-to-zero blocks, *'Logic And'* and *'Logic Or'* blocks. For the full library, see Appendix A.2 on page 141. This library of theories of *Simulink* blocks could be extended in order to accommodate a bigger class of models, including the ones that require non-scalar blocks and signals to be represented graphically as a *Simulink* model.

Step 2

In order to express verification goals graphically in *Simulink*, a custom 'Require' block is created, as depicted in Figure 3.2. This block has two input ports, denoted as *'Pre_in'* and *'Post_in'*, which must be of Boolean type. By connecting signal *'in1'* to the port *'Pre_in'* and signal *'in2'* to the port *'Post_in'*, we add a verification condition of the form:

$$\forall k,\ in1 = true \to in2 = true \tag{3.3}$$

i.e. for all time steps k, if a pre-condition *'Pre_in'* holds, then the post-condition *'Post_in'* must hold as well.

Output ports *Pre_out* and *Post_out* just replicate the respective input ports, while the port

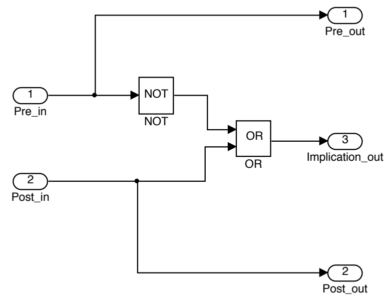


Fig. 3.2 Custom 'Require' *Simulink* block used to add verification conditions to the model.

Implication_out has value $\neg$ *Pre_in* $\bigvee$ *Post_in*, which is equivalent to *Pre_in* $\rightarrow$ *Post_in*. Output ports are only used for checking their respective values in a *Simulink* simulation, while input ports are used in the automatic translation process in *step 3* in order to add specified verification goals to a *Why3* theory.

For example, consider a discrete time first order system of the form $x(k+1) = Kx(k)$ depicted in Figure 3.3 on page 51. In order to argue about the stability of this system, we implemented a Lyapunov function $V(x) = x^2$ graphically in the *Simulink* diagram. Then each of the 'Require' blocks represents a verification condition which, if proven, guarantees asymptotic stability:

1. 'Req1:' $\forall k,\ x(k) \neq 0 \rightarrow V(x(k+1)) - V(x(k)) < 0$, i.e. at each time step k, if the state of the system is not at the origin, the Lyapunov function must be decreasing.
2. 'Req2:' $\forall k,\ x(k) \neq 0 \rightarrow V(x(k)) > 0$, i.e. at each time step k, if the state of the system is not at the origin, the Lyapunov function must be positive.
3. 'Req3:' $\forall k,\ x(k) = 0 \rightarrow V(x(k)) = 0$, i.e. at each time step k, if the state of the system is at the origin, the Lyapunov function must be zero.

Step 3

In this step, a *MATLAB* function (for full code of the function, see Appendix A.3 on page 145) automatically translates a *Simulink* model like the one in Figure 3.3 into a corresponding *Why3* theory. When called, this translation function automatically performs the following steps:

1. Identification of 'Require' blocks in the *Simulink* model. This step is needed because 'Require' blocks represent verification goals and not the actual functionality of the control system.

2. Standard *Why3* theories used in theories for scalar *Simulink* blocks (such as theories for integers, real numbers and Boolean expressions) are imported. For the *Why3* standard library theory `RealInfix` for real numbers (and other *Why3* standard library theories it depends on), see Appendix A.1 on page 137.

3. For all the blocks in the diagram (except the 'Require' ones), every outgoing signal is identified, named and translated to an appropriate uninterpreted function. For example, an *n-th* output signal of the block named *block_name* will be translated to the function:

   ```
   function <block_name>_p<n> int : <real/bool>
   ```

   where `int` and `real/bool` indicate the type of the signal (integer, real and Boolean, respectively).

4. For each block in the model (except the 'Require' ones), an appropriate *Why3* theory for that scalar block is cloned from the file 'Theories.why' with an appropriate parametrisation of input and output functions, gains, etc. For example, consider a 'Gain' block named `Neg` in Figure 3.3 with a gain of -1. It is translated to

   ```
   clone Theories.Gain as Neg with function in1 = vx_old_p1,
   function out1 = neg_p1
   axiom Neg_gain: Neg.gain =  -.1.0
   ```

   where `-.1.0` represents the value of the gain (i.e., -1).

5. For each 'Require' block, a verification goal is created. Suppose that the signals named *prec* and *post* are connected to the input ports *'Pre_in'* and *'Post_in'* of the 'Require' block, respectively. Then a verification goal of the form

   ```
   goal <name_of_goal> : forall k: int. prec k = True -> post k = True
   ```

   is added. Verification goal of this form represents the requirement that, for all time steps k ($k = 0, 1, 2, \ldots$), if some precondition *prec* holds (`prec k = True`), then post-condition *post* is true as well (`post k = True`).

For example, the translation function produces the following *Why3* theory for the *Simulink* diagram shown in Figure 3.3:

```
theory Lyapunov_stability
   use import int.Int
   use import real.RealInfix
   use import bool.Bool

   function desc_grad_p1 int:bool
   function difference_p1 int:real
```

```
   function neg_p1 int:real
   function not_zero_x_p1 int:bool
   function vx_p1 int:real
   function vx_old_p1 int:real
   function vx_pos_p1 int:bool
   function vx_zero_p1 int:bool
   function x_p1 int:real
   function x_not_zero_p1 int:bool
   function x_old_p1 int:real
   function x_zero_p1 int:bool

   clone Theories.LessThanZero as Desc_grad with function in1 = difference_p1,
   function out1 = desc_grad_p1

   clone Theories.Sum as Difference with function in1 = neg_p1, function in2 = vx_p1,
   function out1 = difference_p1

   clone Theories.Gain as Neg with function in1 = vx_old_p1, function out1 = neg_p1
   axiom Neg_gain: Neg.gain =  -.1.0

   clone Theories.NotEqualToZero as Not_zero_x with function in1 = x_p1, function out1 =
   not_zero_x_p1

   clone Theories.Product as Vx with function in1 = x_p1, function in2 = x_p1, function out1
   = vx_p1

   clone Theories.Product as Vx_old with function in1 = x_old_p1, function in2 = x_old_p1,
   function out1 = vx_old_p1

   clone Theories.GreaterThanZero as Vx_pos with function in1 = vx_p1, function out1 =
   vx_pos_p1

   clone Theories.EqualToZero as Vx_zero with function in1 = vx_p1, function out1 =
   vx_zero_p1

   clone Theories.Gain as X with function in1 = x_old_p1, function out1 = x_p1
   axiom X_gain: X.gain >. -.1.0 /\ X.gain <. 1.0

   clone Theories.NotEqualToZero as X_not_zero with function in1 = x_p1, function out1 =
   x_not_zero_p1

   clone Theories.UnitDelay as X_old with function in1 = x_p1, function out1 = x_old_p1

   clone Theories.EqualToZero as X_zero with function in1 = x_p1, function out1 = x_zero_p1

   goal G1 : forall k: int. not_zero_x_p1 k = True -> desc_grad_p1 k = True
   goal G2 : forall k: int. x_not_zero_p1 k = True -> vx_pos_p1 k = True
   goal G3 : forall k: int. x_zero_p1 k = True -> vx_zero_p1 k = True
end
```

Step 4

The created *Why3* theory is sent to one of the SMT solvers via a *Why3* tool. SMT solvers automatically discharge verification conditions in the input theory, thus checking whether the system satisfies required properties at all time steps. ■

In Sections 3.1.2.1 and 3.1.2.2, we apply this *Why3* verification framework to simple systems of the form $x(k+1) = Kx(k)$ with $x(k) \in \mathbb{R}$ and $x(k) \in \mathbb{R}^2$, respectively.

3.1.2.1 Computational example: first order system

Consider a first order system of the form

$$x(k+1) = Kx(k),\ k = 0,1,2,\ \ldots \tag{3.4}$$

with $x(k) \in \mathbb{R}$, $K \in \mathbb{R}$, together with appropriate scalar *Simulink* blocks needed to implement the Lyapunov function $V(x) = x^2$ and 'Require' blocks for adding verification conditions as shown in Figure 3.3, with the corresponding *Why3* theory `Lyapunov_stability` given on page 48. The three verification goals in theory `Lyapunov_stability` represent the following verification conditions expressed in terms of the Lyapunov function $V(x) = x^2$:

- `goal G1`: $\forall\, x(k) \neq 0,\ V(x(k+1)) - V(x(k)) < 0$.
- `goal G2`: $\forall\, x(k) \neq 0,\ V(x(k)) > 0$.
- `goal G3`: $V(x(k)) = 0$ for $x(k) = 0$.

If all of these verification goals can be discharged by some SMT solver, then asymptotic stability of the system is verified.

Prover	`goal G1`	`goal G2`	`goal G3`
Alt-Ergo	Unknown (realised under 50s)	Proven (under 0.1s)	Proven (under 0.1s)
CVC3	Unknown (realised under 120s)	Proven (under 0.1s)	Proven (under 0.1s)
Z3	Proven (under 0.5s)	Proven (under 0.1s)	Proven (under 0.1s)

Table 3.1 Verification results with $|K| < 1$.

Suppose $|K| < 1$ in the system (3.4). In the *Why3* theory `Lyapunov_stability` representing this system, this assumption is represented by the following axiom on the gain value:

```
axiom X_gain: X.gain >. -.1.0 /\ X.gain <. 1.0
```

i.e., in this *Why3* theory, instead of giving a specific value of K, we just assume that $|K| < 1$.

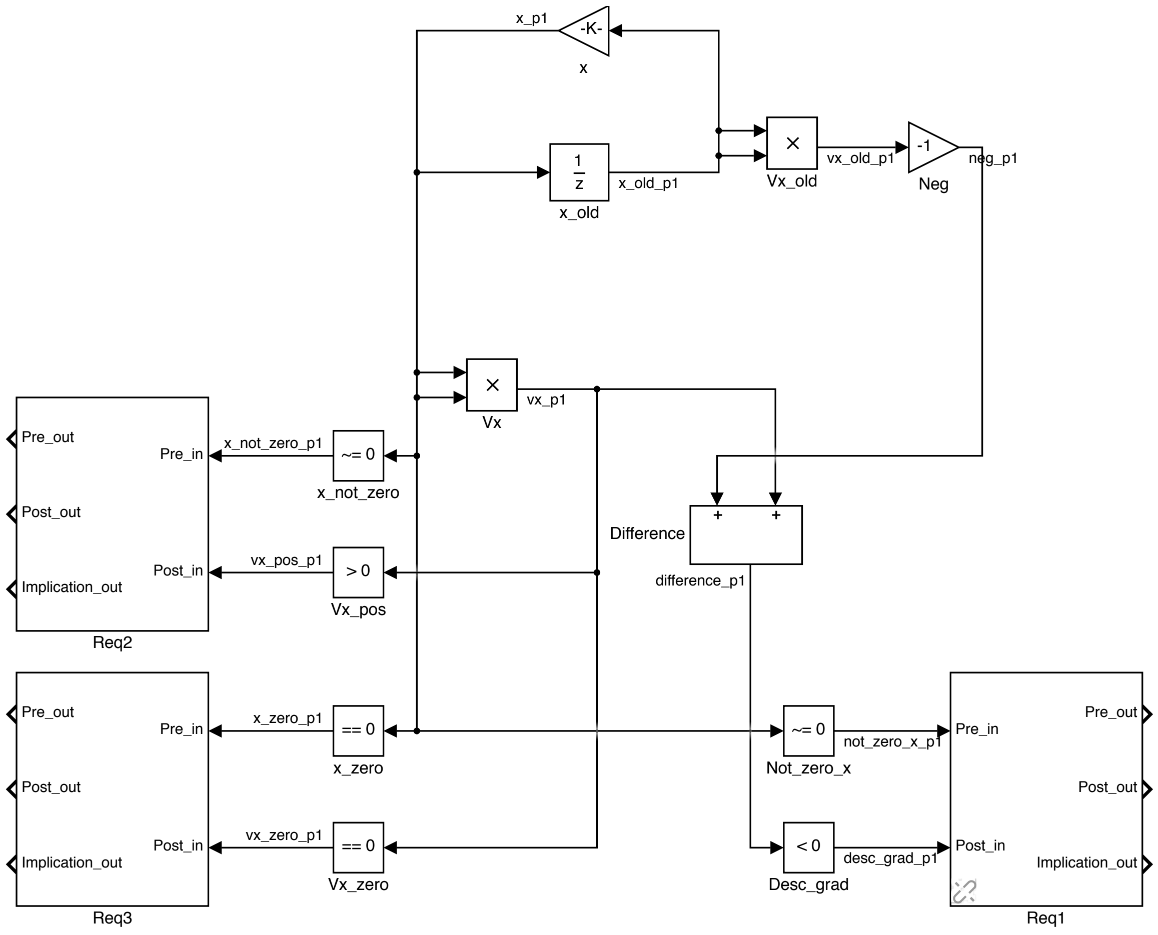


Fig. 3.3 Graphical implementation of a first order system $x(k+1) = Kx(k)$ in *Simulink*, together with 'Require' blocks *Req1*, *Req2* and *Req3* representing verification criteria.

MATLAB translation script, given the *Simulink* diagram illustrated in Figure 3.3 as an input, successfully produces a corresponding *Why3* theory `Lyapunov_stability` in a fraction of a second (*step 3* in the verification framework). Results of verification obtained when `Lyapunov_stability` *Why3* theory is passed to SMT provers Alt-Ergo, CVC3 and Z3 provers (*step 4* in the verification framework) are summarised in Table 3.1. Prover Z3 is capable of discharging all three goals, hence proving the asymptotic stability of the system (3.4). Alt-Ergo and CVC3 are capable of dealing with goals `G2` and `G3`, but cannot discharge goal `G1` (i.e., provers return proof-status 'Unknown') because of their inability to deal with universal quantification and non-linear arithmetic inequalities (since the goal `G1` becomes $\forall x(k) \neq 0$, $x^2(k)(K^2-1) < 0$).

On the other hand, suppose $|K| \geq 1$. This assumption in the *Why3* theory would be represented as:

```
axiom X_gain: X.gain <=. -.1.0 \/ X.gain >=. 1.0
```

With this assumption in place, all provers (Alt-Ergo, CVC3 and Z3) still prove goals `G2` and `G3`, as expected. While trying to prove the goal `G1`, CVC3 runs out of time (given a time limit of one hour), while Alt-Ergo and Z3 are not capable of discharging this particular goal because of its mathematical structure (proof-status 'Unknown' returned by the provers). Hence, these provers are not capable of showing that the system (3.4) with $|K| \geq 1$ does not satisfy the Lyapunov stability requirement represented by goal `G1`.

Now suppose the value of K in (3.4) is left unspecified, i.e. no particular numerical value is assigned to K and no bounds on it are introduced in a form of an axiom in the `Lyapunov_stability` *Why3* theory. In this case, goals `G2` and `G3` are still discharged by all SMT provers. Alt-Ergo and CVC3 are incapable of dealing with goal `G1` (proof-status 'Unknown' returned), while Z3 runs out of time (the time limit being 6 hours) trying to discharge this goal.

A conclusion regarding these results, and a way forward using quantifier elimination based verification approach, is presented in Section 3.1.3.

3.1.2.2 Computational example: second order system

Similarly to the Section 3.1.2.1, consider a second order system of the form

$$x(k+1) = Ax(k),\ k = 0,1,2,\ \ldots, \tag{3.5}$$

where $x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$ and $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$. Figure 3.4 shows this system implemented using only scalar *Simulink* blocks and scalar signals, together with other scalar blocks needed to implement a Lyapunov function $V(x) = x^T Px$ and 'Require' blocks for adding verification conditions. Verification goals are analogous to the ones in Section 3.1.2.1, the only differ-

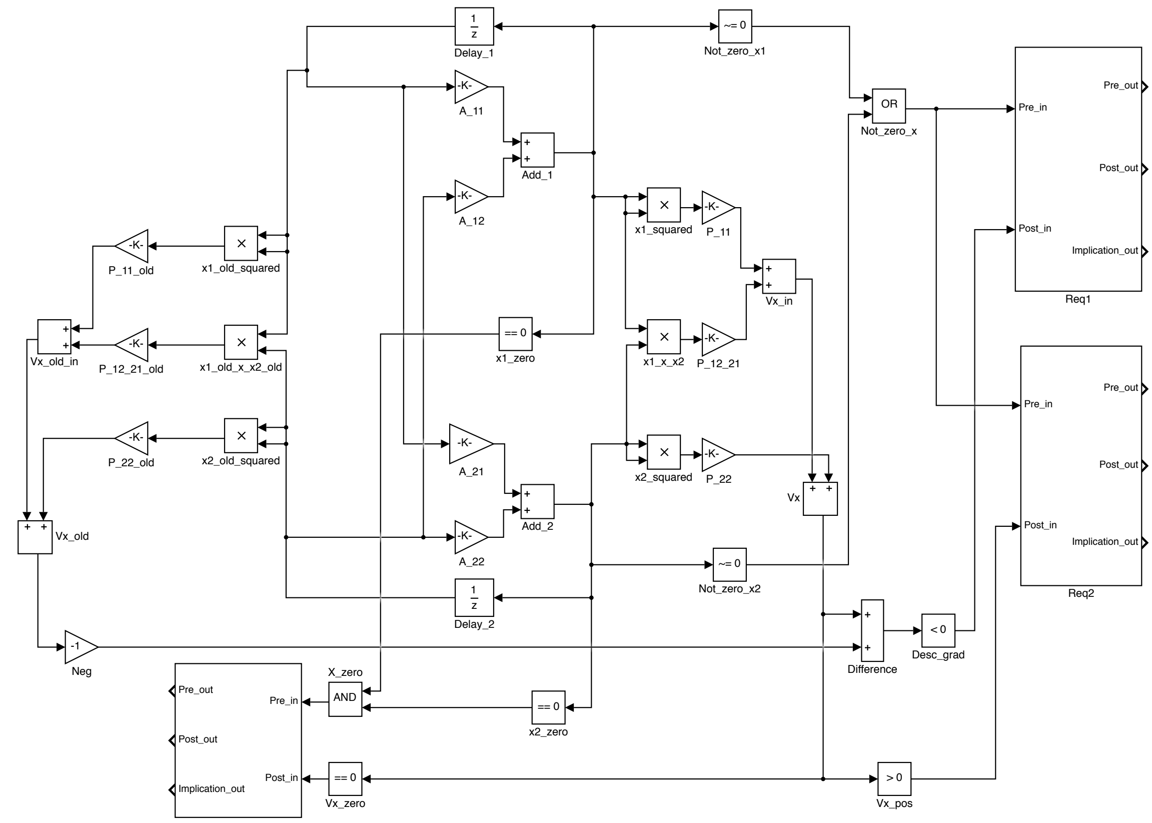


Fig. 3.4 Graphical implementation of a second order system $x(k+1) = Ax(k)$ in *Simulink* using only scalar blocks and signals, together with 'Require' blocks *Req1*, *Req2* and *Req3* representing verification criteria.

ence being that the Lyapunov function now is of the form $V(x) = x^T Px$ for some positive definite matrix P.

As before, the *MATLAB* translation script, given the *Simulink* diagram illustrated in Figure 3.4 as an input, successfully produces a corresponding *Why3* theory in a fraction of a second. Results of verification obtained for several systems when this *Why3* theory is passed to SMT provers are shown in Table 3.2, where the matrix Q is the following matrix in the discrete Lyapunov equation:

$$A^T PA - P = -Q. \tag{3.6}$$

In the first, second and fourth cases, CVC3 and Z3 are capable of discharging all goals under four seconds, hence proving the asymptotic stability of the system of the form (3.5). In the third case, goal `G1` is proven only by Z3, goal `G2` is proven only by Alt-Ergo while goal `G3` is discharged by all provers — therefore, this second order system is proved to be asymptotically stable as well.

Case #	1	2	3	4
A	$\begin{bmatrix}0.5 & 0\\ 0 & 0.6\end{bmatrix}$	$\begin{bmatrix}0.5 & 0\\ 0 & -0.9999\end{bmatrix}$	$\begin{bmatrix}0.2 & 4\\ 0 & 0.5\end{bmatrix}$	$\begin{bmatrix}6 & -3\\ 12 & -6\end{bmatrix}$
P	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}9 & 8\\ 8 & 240\end{bmatrix}$	$\begin{bmatrix}761 & -377\\ -375 & 191\end{bmatrix}$
Q	$\begin{bmatrix}0.75 & 0\\ 0 & 0.64\end{bmatrix}$	$\begin{bmatrix}0.75 & 0\\ 0 & 0.0002\end{bmatrix}$	$\begin{bmatrix}8.64 & 0\\ 0 & 4\end{bmatrix}$	$\begin{bmatrix}5 & 1\\ 3 & 2\end{bmatrix}$
Alt-Ergo	`G1`: unknown `G2`: proven (under 2s) `G3`: proven (under 40s)	`G1`: unknown `G2`: proven (under 1.6s) `G3`: proven (under 32s)	`G1`: unknown `G2`: proven (under 47s) `G3`: proven (under 115s)	`G1`: unknown `G2` and `G3`: ran out of time (1h)
CVC3	All goals proven (under 2.5s)	All goals proven (under 2.1s)	`G1` and `G2`: unknown `G3` - proven (under 5s)	All goals proven (under 4s)
Z3	All goals proven (under 0.1s)	All goals proven (under 0.1s)	`G1` and `G3`: proven (under 5s) `G2` - ran out of time (1h)	All goals proven (under 1s)

Table 3.2 Verification results for various A and P values.

On the other hand, consider the system

$$x(k+1) = \begin{bmatrix}0.2 & 4\\ 0 & 0.5\end{bmatrix} x(k) \tag{3.7}$$

which was proven to be asymptotically stable by using the Lyapunov function

$$V(x) = x^T \begin{bmatrix}9 & 8\\ 8 & 240\end{bmatrix} x \tag{3.8}$$

(as shown in Table 3.2). In Table 3.3, we consider the same system with different values of matrix P for the Lyapunov function (all of which are positive definite). In this table, the matrix Q is as described by the equation (3.6).

Only in the first case the provers were able to collectively discharge all three goals (i.e., each one of the three goals was discharged by at least one of the provers), consequently proving asymptotic stability, while in the second, third and fourth cases, none of the provers

Case #	1	2	3	4
P	$\begin{bmatrix} 9 & 8 \\ 8 & 240 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 28 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 28 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 1 & 36 \end{bmatrix}$
Q	$\begin{bmatrix} 8.64 & 0 \\ 0 & 4 \end{bmatrix}$	$\begin{bmatrix} 0.96 & -0.8 \\ -0.8 & 5 \end{bmatrix}$	$\begin{bmatrix} 0.96 & 0.1 \\ 0.1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0.96 & 1 \\ 0.1 & 5 \end{bmatrix}$
Alt-Ergo	G1: unknown G2: proven (under 47s) G3: proven (under 115s)	G1: unknown G2: proven (under 2s) G3: proven (under 33s)	G1: unknown G2: proven (under 47s) G3: proven (under 89s)	G1: unknown G2: proven (under 48s) G3: proven (under 120s)
CVC3	G1 and G2: ran out of time (1h) G3 - proven (under 5s)	G1: ran out of time (1h) G2 and G3: proven (under 5s)	G1 and G2: unknown G3: proven (under 5s)	G1 and G2: unknown G3: proven (under 5s)
Z3	G1 and G3: proven (under 5s) G2 - ran out of time (1h)	G1: ran out of time (1h) G2: proven (under 30s) G3: proven (under 5s)	G1 and G2: ran out of time (1h) G3: proven (under 5s)	G1 and G2: ran out of time (1h) G3: proven (under 5s)

Table 3.3 Verification of a system (3.7) with various values of P for Lyapunov function $V(x) = x^T Px$.

could deal with the goal G1 despite the suitable choice of P matrix. Additionally, in the first and second cases in Table 3.3, CVC3 runs out of time while trying to discharge G1, but stops before the time limit in third and fourth cases because it detects non-linear arithmetic it is not capable of dealing with. Moreover, notice that when the matrix Q is diagonal, asymptotic stability is always proven. Conversely, when Q is not diagonal, asymptotic stability only gets proven for a trivial system $A = \begin{bmatrix} 6 & -3 \\ 12 & -6 \end{bmatrix}$ (see Table 3.2) where the matrix A has a single eigenvalue with an algebraic multiplicity of 2.

Similarly, instead of using a specific matrix P, using just the fact that the matrix P must be positive definite (which is expressed by adding the following axiom in the corresponding *Why3* theory):

```
axiom P_gain: forall x1 x2: real.
              (x1 >. 0 /\ x2 >. 0) \/ (x1 <. 0 /\ x2 <. 0) \/
              (x1 >. 0 /\ x2 <. 0) \/ (x1 <. 0 /\ x2 >. 0)
              ->
              P_11.gain *. x1 *. x1 +.
              P_12_21.gain *. x1 *. x2 +.
              P_22.gain *. x2 *. x2 >. 0.0
```

leads to all provers running out of time (time limit being six hours) while trying to deal with the goal `G1`.

A conclusion regarding these results, and a way forward using quantifier elimination based verification approach, is presented in Section 3.1.3.

3.1.3 Conclusions

In principle, the *Why3* verification framework developed in Section 3.1.2 could be applied to any control system implemented in *Simulink* as long as a corresponding *Why3* theory is developed for each type of *Simulink* block used in this implementation (including *Simulink* blocks used to implement verification criteria). For a current library of such *Why3* theories, see Appendix A.2 on page 141.

Unfortunately, as illustrated in Sections 3.1.2.1 and 3.1.2.1, we run into computational issues (SMT provers running out of time or returning proof-status 'Unknown' indicating their inability to proceed) even when trying to prove a simple property of global asymptotic stability for systems of the form $x(k+1) = Kx(k)$ with $x(k) \in \mathbb{R}$ or $x(k) \in \mathbb{R}^2$.

Expressing the system of interest as a *Simulink* diagram and then converting it to an equivalent *Why3* theory to prove a simple property such as asymptotic stability introduces a lot of variables (which correspond to signals in the *Simulink* diagram and are declared as uninterpreted functions in the *Why3* theory — see `Lyapunov_stability` on page 48 for an example). This, at the end of the day, just represents the verification problem in a complex way while at the same time not necessarily making it easier to analyse more complicated robustness properties.

On the other hand, many analysis and synthesis problems in control theory can be represented as a quantifier elimination problem of the form (2.7) on page 16. This approach requires fewer variables to express verification requirement of interest than a *Why3* verification framework (since, unlike *Why3* verification framework, it does not have to explicitly deal with a multitude of variables coming from signals in the *Simulink* diagram). For example, for a given positive definite matrix P and system A, goal `G1` can be expressed as a quantifier elimination problem of the form:

$$\forall x_1, x_2,\ x_1 \neq 0 \vee x_2 \neq 0 : \begin{bmatrix} x_1 & x_2 \end{bmatrix} Q \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} > 0. \tag{3.9}$$

As expected, quantifier elimination algorithms show that (3.9) is equivalent to *true* for all choices of A and P represented in Tables 3.2 and 3.3.

Similarly, for a given system A and general matrix $P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}$, the asymptotic stability can be proven by eliminating quantifiers from the following first order formula:

$$\exists P_{11}, P_{12}, P_{21}, P_{22} : \left(\forall x_1, x_2,\ x_1 \neq 0 \vee x_2 \neq 0 : \left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} P \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} > 0 \right) \wedge \left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} Q \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} > 0 \right) \right). \tag{3.10}$$

Again, for all the systems A from Table 3.2, quantifier elimination algorithm shows that (3.10) is equivalent to $true$, as expected.

Hence, from computational perspective, we see that using quantifier elimination algorithms to prove asymptotic stability of the system is more practical than trying to do so via the *Why3* based framework considered here. Therefore, in the following sections, we focus on checking various criteria by expressing it as a quantifier elimination problem first.

3.2 Exclusion regions

In this section, we consider several verification criteria that can be expressed in terms of exclusion regions. In order for the robustness specification expressed in terms of an exclusion zone to hold, the graph of the open loop transfer function $L(j\omega)$ representing the system must not enter the specified region. To check whether or not this is the case, the avoidance condition is converted by us to an equivalent quantifier elimination problem. In Section 3.2.1, we will discuss the M-circle exclusion region while Section 3.2.2 will consider exclusion regions whose avoidance guarantees that sufficient simultaneous gain and phase margins are attained.

3.2.1 M-circle exclusion region

Consider an M-circle exclusion region (for reference, see Levine (1996), section 10.3.8), which is defined as a bound on the magnitude of the complementary sensitivity function

$$|T(j\omega)| = \left| \frac{L(j\omega)}{1 + L(j\omega)} \right| \leq M, \tag{3.11}$$

where $L(s)$ is an open loop transfer function and $M > 0$ is a constant. Let $L(j\omega) = x + iy$ (i.e., $x = \mathrm{Re}\,(L(j\omega))$ and $y = \mathrm{Im}\,(L(j\omega))$). Then the inequality (3.11) can be written as

$$\left(x + \frac{M^2}{M^2 - 1} \right)^2 + y^2 \leq \left(\frac{M}{M^2 - 1} \right)^2, \tag{3.12}$$

i.e. an M-circle exclusion region is a disk in a Nyquist plane with a centre $x_c = \frac{M^2}{1-M^2}, y_c = 0$ and a radius $r = \left|\frac{M}{M^2-1}\right|$.

Some important properties of M-circles are:

- If the open loop transfer function $L(s)$ does not enter the M-circle of value M, then the complementary sensitivity gain plot will stay below $20\log_{10} M$.
- If $M_1 > M_2 > 1$, then the M_2-circle contains the M_1-circle.
- For $M = 1$, the M-circle reduces to the vertical line $x = -\frac{1}{2}$.
- If M converges to 1 from below, the M-circle converges to the right half plane given by $x \geq -\frac{1}{2}$; if M converges to 1 from above, then the M-circle converges to the left half plane given by $x \leq -\frac{1}{2}$.

In order to prove with a quantifier elimination (QE) algorithm that the open loop transfer function $L(j\omega)$ does not enter the M-circle exclusion region, we have to show that:

$$\forall \omega : |L(j\omega) - x_c| > r. \tag{3.13}$$

Hence, the QE algorithm has to discharge the following first order formula:

$$\forall \omega : (Re(L(j\omega)) - x_c)^2 + Im(L(j\omega))^2 > r^2. \tag{3.14}$$

Suppose $L(j\omega)$ is a rational transfer function of the form

$$L(j\omega) = \frac{a(\omega) + i \cdot b(\omega)}{c(\omega)}, \tag{3.15}$$

where $a(\omega), b(\omega)$ and $c(\omega)$ are real functions of ω. Then the requirement (3.14) reduces to the condition that a particular polynomial is positive for all frequencies ω:

$$\forall \omega : (a(\omega) - x_c c(\omega))^2 + b(\omega)^2 - r^2 c(\omega)^2 > 0. \tag{3.16}$$

If $L(s)$ is a discrete-time transfer function (as is the case for the backup flight control law discussed in Section 3.6.1), it has to be evaluated at $s = e^{j\theta} = \cos(\theta) + j \cdot \sin(\theta)$. In this case, the polynomial in (3.16) becomes a trigonometric polynomial $f(\cos(\theta), \sin(\theta))$ in terms of θ:

$$\forall \theta : (0 \leq \theta \wedge \theta < 2\pi) \Rightarrow f(\cos(\theta), \sin(\theta)) > 0. \tag{3.17}$$

In order to deal with this, we make a substitution $X = \cos(\theta)$, $Y = \sin(\theta)$ and require the quantifier elimination algorithm to discharge:

$$\forall X, Y : (X^2 + Y^2 = 1) \Rightarrow f(X, Y) > 0. \tag{3.18}$$

This is indeed what we do in Section 3.6.2 when trying to verify criteria of interest for the discrete-time backup flight control law discussed in Section 3.6.1.

3.2.2 Exclusion regions for simultaneous gain and phase margins

Suppose we want to verify that the system, described by an open loop transfer function $L(s)$, has simultaneous gain and phase margins G_m and P_m, respectively. This can be achieved by making sure that $L(j\omega)$ does not enter a chosen exclusion region around the critical point $(-180^\circ,\ 0\,\mathrm{dB})$ in the Nichols plane. For example, a hexagonal exclusion region depicted in Figure 3.5 (a) guarantees simultaneous gain and phase margins of 3 dB and 30°. Note that the hexagonal exclusion region is somewhat conservative, since non-convex exclusion regions around the critical point $(-180^\circ,\ 0\,\mathrm{dB})$ in the Nichols plane could be used to represent the simultaneous gain and phase margin requirement.

While, in principle, proving avoidance of the hexagonal exclusion region in the Nichols plane by using *MetiTarski* theorem prover for real-valued functions is possible, for non-trivial examples, it is almost always computationally intractable. This intractability arises from having to repeatedly approximate real-valued functions with appropriate polynomial bounds. Hence, we overbound this region by an ellipse:

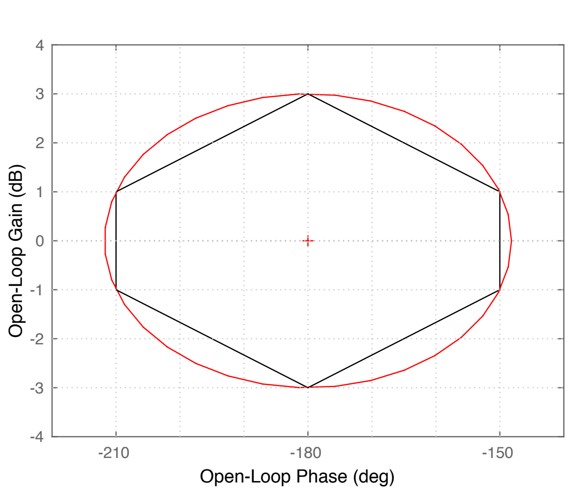
$$\frac{|L(j\omega)|_{dB}^2}{G_m^2} + \frac{(\angle L(j\omega) + 180)^2}{P_m^2} = 1. \tag{3.19}$$

This obviously introduces some additional conservatism, because the uncertainty region is larger than the strictly required exclusion zone.
As was shown in Deodhare and Patel (1998), if we let G_m and P_m in (3.19) be

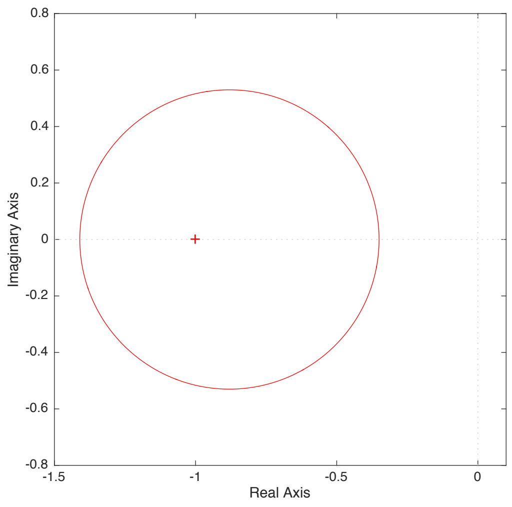
$$G_m = 20\log_{10}(a + r), \tag{3.20}$$

$$P_m = \cos^{-1}\left(\frac{a^2 - r^2 + 1}{2a}\right), \tag{3.21}$$

then this elliptical Nichols exclusion region can be non-conservatively overbounded by a circular exclusion region in the Nyquist plane with a center $(-a,\ 0)$ and radius r with



(a) Exclusion regions in the Nichols plane around the critical point $(-180°,\ 0\,\text{dB})$.



(b) Circular overbounding ($a = 0.88,\ r = 0.53$) in the Nyquist plane of the elliptical Nichols exclusion region shown in figure (a).

Fig. 3.5 Exclusion regions.

$a > 0$, $a > r$ and $r > |a-1|$ (i.e. critical point $(-1,\ 0)$ in the Nyquist plane is contained inside the circle), as can be seen in Figure 3.5 (b). Therefore, for example, if the open loop frequency response $L(j\omega)$ does not enter the circular exclusion region shown in Figure 3.5 (b), it implies that $L(j\omega)$ does not enter the elliptical Nichols exclusion region depicted in Figure 3.5 (a), and therefore the system has a guaranteed simultaneous gain and phase margin of 3 dB and 32°.

Deodhare and Patel (1998) showed that $L(j\omega)$ not entering the described circular exclusion region is equivalent to avoidance of $(-1,0)$ critical point in the Nyquist plane by an open loop frequency response $\hat{L}(j\omega)$ of all perturbed systems of the form

$$\hat{L}(s) = L(s)(a+\Delta),\ ||\Delta||_\infty \leq r, \tag{3.22}$$

which can obviously be rewritten as:

$$\hat{L}(s) = aL(s)\left(1+\frac{r}{a}\Delta\right),\ ||\Delta||_\infty \leq 1. \tag{3.23}$$

Equation (3.23) can be interpreted as a multiplicative uncertainty on the scaled system. Via the small gain theorem, this can be shown to be equivalent to an upper bound on the complementary sensitivity function for a scaled plant:

$$\left\| \frac{AL(j\omega)}{1+AL(j\omega)} \right\|_\infty < \frac{a}{r}, \text{ where } A = \frac{a}{a^2-r^2}. \tag{3.24}$$

Note that for $A = 1$, this reduces to an M-circle criterion described in Section 3.2.1.

Suppose $L(j\omega)$ is expressed as in equation (3.15). Then, in summary, if a quantifier elimination algorithm discharges the following first order formula

$$\forall \omega : (a(\omega) + ac(\omega))^2 + b(\omega)^2 - r^2 c(\omega)^2 > 0, \tag{3.25}$$

then the following claims can be made:

- The system described by an open loop frequency response $L(j\omega)$ has a simultaneous gain and phase margin as shown in equations (3.20) and (3.21), respectively.
- All uncertain systems with an open loop transfer function (3.23) are closed-loop stable.

Note that robust stability requires not only staying outside the exclusion region, but also encircling it the correct number of times (0 if open-loop stable, according to the Nyquist stability criterion). It is enough to check closed-loop stability with the nominal system $L(s)$.

This simultaneous gain and phase margin verification condition will be cleared for the backup flight control law discussed in Section 3.6.1 at given flight points (see Section 3.6.2) and throughout the whole flight envelope (see Section 3.6.3).

3.3 Structured singular value μ computation as a quantifier elimination problem

In this section, a quantifier elimination based approach to the computation of the structured singular value μ problem is considered. Firstly, in Section 3.3.1, relevant definitions and standard ways of computing μ are discussed. Then, in Sections 3.3.2 and 3.3.3, approaches of expressing the μ computation problem as a quantifier elimination one are introduced, together with a comparison of our method to the standard ones discussed in Section 3.3.1. Finally, in Section 3.3.4, some computational examples illustrating application of quantifier elimination to structured singular value μ computation are given.

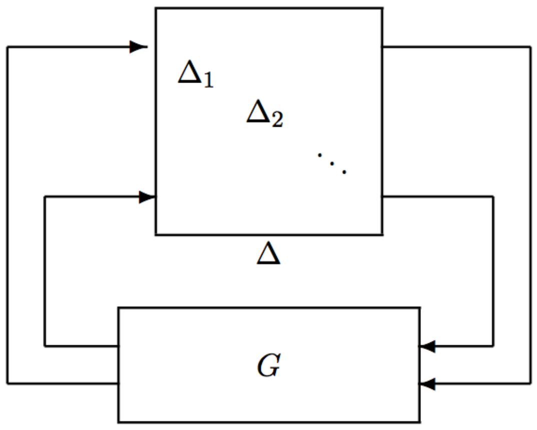


Fig. 3.6 Feedback system with a generalised plant G and uncertainty structure Δ.

3.3.1 Introduction

Consider a feedback interconnection as illustrated in Figure 3.6, where G is a closed-loop stable system and Δ is a structured norm-bounded uncertainty of the form

$$\boldsymbol{\Delta}(\delta) = \{\Delta : \Delta = \text{block diag}(\Delta_1(s), \dots, \Delta_n(s)), \Delta_i \in H_\infty^{r_i \times r_i}, ||\Delta_i||_\infty \leq \delta\} \subset H_\infty^{m \times m}, \quad (3.26)$$

where $H_\infty^{r_i \times r_i}$ denotes the set of stable $r_i \times r_i$ transfer functions with a finite H_∞ norm and $m = \sum_{i=1}^{n} r_i$. This kind of representation raises an obvious question: is the system G robustly stable under the structured uncertainty $\Delta \in \boldsymbol{\Delta}(\delta)$? In order to answer this question non-conservatively, a measure called *structured singular value* μ was introduced in Doyle (1982) that takes into account the structure of the uncertainty $\Delta \in \boldsymbol{\Delta}(\delta)$. It is defined as

$$\mu(G) = \frac{1}{\min_{\Delta \in \boldsymbol{\Delta}(\delta)} \{\overline{\sigma}(\Delta) \text{ s.t. } \det(I - G\Delta) = 0\}}, \quad (3.27)$$

unless no $\Delta \in \boldsymbol{\Delta}(\delta)$ makes $(I - G\Delta)$ singular, in which case $\mu(G) = 0$. In (3.27), $\overline{\sigma}(\Delta)$ denotes the largest singular value of the matrix Δ — for more details, see Chapter 10 in Zhou and Doyle (1998).

The system G is robustly stable $\forall \Delta \in \boldsymbol{\Delta}(\delta)$ if and only if:

$$\forall \omega, \mu(G(j\omega)) < \frac{1}{\delta}. \quad (3.28)$$

The exact computation of the structured singular value μ (3.27) is known to be NP-hard for the general case (Braatz et al., 1994). Hence, literature focuses on developing methods for computing both lower and upper bounds on μ instead. The standard way to compute the upper bound on μ in polynomial time was developed in Young et al. (1995), and it proceeds as follows.

Proposition 2 *(Young et al., 1995)*
Consider a feedback interconnection as illustrated in Figure 3.6, with G being a closed-loop stable system and Δ being a structured norm-bounded uncertainty of the form (3.26).

Let β_i be a positive scalar, ω_i be some given frequency and D^* denote the complex conjugate transpose of the matrix D. Then, if there exist some scaling matrices $D_1(\omega_i) \in \mathscr{D}_1$ and $D_2(\omega_i) \in \mathscr{D}_2$, where $\mathscr{D}_1$ and $\mathscr{D}_2$ are the sets reflecting the block-diagonal and real/complex structure of $\Delta(\delta)$

$$\mathscr{D}_1 = \{D_1 \in \mathbb{C}^{m \times m}, D_1 = D_1^* > 0 : \forall \Delta \in \Delta(\delta), D_1\Delta = \Delta D_1\},$$
$$\mathscr{D}_2 = \{D_2 \in \mathbb{C}^{m \times m}, D_2 = D_2^* : \forall \Delta \in \Delta(\delta), D_2\Delta = \Delta^* D_2\},$$

such that

$$\overline{\sigma}\left((I + D_2(\omega_i))^{-\frac{1}{4}} \left(\frac{D_1(\omega_i)G(j\omega_i)D_1(\omega_i)^{-1}}{\beta_i} - jD_2(\omega_i) \right) (I + D_2(\omega_i))^{-\frac{1}{4}} \right) \leq 1, \quad (3.29)$$

then $\mu(G(j\omega_i)) \leq \beta_i$. ■

The problem of minimizing β_i for each of frequencies ω_i can be solved by using a Linear Matrix Inequality (LMI) solver or a gradient descent algorithm — for more details, see Young et al. (1995). Usually, the calculation of this upper bound is performed on a finite frequency grid (see implementation of the function `mussv` in the MATLAB Robust Control Toolbox The MathWorks Inc. (2012a)).

Now consider $\Delta \in \Delta(\delta)$ (3.26) with all the uncertainties being real parametric ones, i.e. $\Delta_i \in \mathbb{R}, i = 1, \ldots, n$. Then the uncertainty structure expressed by (3.26) reduces to

$$\Delta_R(\delta) = \{\Delta : \Delta = \mathrm{diag}(\Delta_1(s), \ldots, \Delta_n(s)), \Delta_i \in \mathbb{R}, |\Delta_i| \leq \delta\}, \quad (3.30)$$

while the structured singular value μ computation reduces to:

$$\mu(G) = \frac{1}{\min_{\Delta \in \Delta_R(\delta)} \{\delta |\det(I - G\Delta) = 0\}}. \quad (3.31)$$

It is worth noting that computation of the so called real-*mu* value (3.31) is also known to be an NP-hard problem (Braatz et al., 1994).

With appropriate scaling on the uncertainty Δ, (3.31) can be expressed as:

$$\mu(G) = \frac{1}{\min_{\Delta \in \Delta_R(1)}\{k \in [0,+\infty) \text{ s.t. } \det(I-kG\Delta)=0\}} = \frac{1}{k_m}. \tag{3.32}$$

As was noted in de Gaston and Safonov (1988), k_m in (3.32) can be interpreted as the lowest value of k such that the image of the unit hypercube

$$D = \{\Delta = (\Delta_1,\ldots,\Delta_n) \subset \mathbb{R}^n \ : \ \Delta_i \in \mathbb{R}, |\Delta_i| \leq 1\} \tag{3.33}$$

under the mapping $\det(I-kG\Delta)$ contains the origin of the complex plane. If, for some fixed value $k_1 \in [0,+\infty)$, the map of the hypercube D into the complex plane under $\det(I-k_1G\Delta)$ does not include the origin, then $k_1 < k_m$ (and, $\mu = \frac{1}{k_m} < \frac{1}{k_1}$ — i.e., an upper bound on μ is obtained). By repeating the mapping $\det(I-kG\Delta)$ for increasingly larger values of k until the origin is just included in the image, one would obtain the exact value of k_m, and consequently, μ.

Computing an exact image of D (3.33) under the map $\det(I-kG\Delta)$ is, in general, a computationally intractable task (de Gaston and Safonov, 1988). On the other hand, computing the convex hull of the image is relatively practical by using the following result from Zadeh and Desoer (1963).

Proposition 3 *(Zadeh and Desoer, 1963), page 476*
Consider the structured uncertainty $\Delta \in \Delta_R(1)$ (3.30)*, with non-repeated parametric uncertainties* $\Delta_1,\ldots,\Delta_n$*. Let* V_i' *be the image (a point in the complex plane* $\mathbb{C}$*) of the vertex* V_i *of the unit hypercube* D (3.33) *under the map* $\det(I-kG\Delta)$ *for some* $k \in [0,+\infty)$*, i.e.:*

$$V_i'(j\omega) = \det(I-kG(j\omega)V_i), \ i = 1,\ldots,2^n. \tag{3.34}$$

Then the image of the unit hypercube D *by the map* $\det(I-kG\Delta)$ *is contained in the convex hull of the set of points* $\left\{V_1',\ldots,\ V_{2^n}'\right\}$*:*

$$S \equiv co\left\{V_1',\ldots,\ V_{2^n}'\right\} \equiv \left\{\sum_{i=1}^{2^n} \alpha_i V_i' \subset \mathbb{C} | : (\forall i, \alpha_i \geq 0) \wedge \sum_{i=1}^{2^n} \alpha_i = 1\right\}. \tag{3.35}$$

■

Hence, finding the smallest value of k for which the set S (3.35) contains the origin provides a lower bound k_L on $k_m = \frac{1}{\mu}$. This motivates an iterative exponential time procedure in de Gaston and Safonov (1988) for computing k_m (and, consequently, μ) to an arbitrary accuracy with a branch and bound scheme. By keeping partitioning the unit hypercube D into smaller ones, and applying the Proposition 3 repeatedly on each one of them, a more and more accurate lower bound k_L on k_m (and, consequently, a more accurate upper bound on the structured singular value $\mu = \frac{1}{k_m} < \frac{1}{k_L}$) is obtained.

Generalisations of the procedure in de Gaston and Safonov (1988) to cases when the structured uncertainty $\Delta \in \Delta_R(1)$ is allowed to have repeated parametric uncertainties $\Delta_1, \ldots, \Delta_n$ were developed in Sideris and de Gaston (1986) and Sideris and Sanchez Pena (1989). Similar procedures based on the branch and bound approach were developed in Balakrishnan et al. (1991), Chang et al. (1991) and Newlin and Young (1997).

In the following two sections, we show how the computation of the structured singular value μ problem can be expressed and tackled as a quantifier elimination one. In Section 3.3.2, we consider a particular type of single-input single-output (SISO) system, and show how computation of the structured singular value μ in that case can be expressed as an M-circle avoidance problem. This is then converted to an equivalent quantifier elimination problem (as discussed in Section 3.2.1). In Section 3.3.3, we show how structured singular value μ computation can be expressed as a quantifier elimination problem, both for systems with norm-bounded (3.26) and parametric (3.30) uncertainties. The differences between quantifier-elimination based approach for μ computation and the standard approaches discussed in this section (and illustrated by Propositions 2 and 3) are discussed in Section 3.3.3 as well.

3.3.2 Expression of SSV μ computation problem as a quantifier elimination problem in SISO case

Consider a SISO system of the form

$$P_1(s) = (1+\Delta) * P_2(s, \Delta_1, \ldots, \Delta_n) \tag{3.36}$$

with $||\Delta||_\infty < r_1$ and $||\Delta_1||_\infty < r_2, \ldots, ||\Delta_n||_\infty < r_{n+1}$. Then (according to the small gain theorem applied to a multiplicative perturbation), if $\forall\, \Delta_1, \ldots, \Delta_n$, $P_2(s, \Delta_1, \ldots, \Delta_n)$ does NOT enter the M-circle exclusion region with $M = 1/r_1$, then $\mu(P_1) < 1$.

For a particular example, consider a continuous nominal system with an open loop transfer function $L_0(s)$ and an uncertain system $L(s)$ of the form

$$L(s) = (1+\Delta_1(s))\,(1+\Delta_2(s))\,L_0(s) = (1+\Delta_1(s))\,L_1(s), \tag{3.37}$$

where:

$||\Delta_1(s)||_\infty < r_1 = \dfrac{1}{M}$ $-$ fictitious multiplicative perturbation used to represent M-circle exclusion region of value M.

$||\Delta_2(s)||_\infty < r_2$ $-$ actual uncertainty on the nominal system $L_0(s)$ representing unmodelled plant dynamics.

Then the following statements are equivalent:

- Structured singular value $\mu_\Delta(L) < 1$.

- The uncertain system $L_1(s) = (1+\Delta_2(s))\,L_0(s)$ does not enter the M-circle of value M for any uncertainty $\Delta_2(s)$ such that $||\Delta_2(s)||_\infty < r_2$. Equivalently, for all such perturbations $\Delta_2(s)$, the magnitude of the complementary sensitivity of the uncertain system $L_1(s)$ will stay below $20\log_{10} M = -20\log_{10} r_1$ on the Bode gain plot. This will be illustrated with an example in Section 3.3.4 (see Figure 3.7).

Suppose $L_0(s)$ is a rational transfer function of the form

$$L_0(j\omega) = \frac{a(\omega) + i \cdot b(\omega)}{c(\omega)}, \tag{3.38}$$

where $a(\omega), b(\omega)$ and $c(\omega)$ are real functions of ω. Then

$$\begin{aligned} L_1(j\omega) &= (1+\Delta_2(j\omega))\,L_0(j\omega) \\ &= (1+R(\omega)+j\cdot S(\omega))\,\frac{a(\omega)+i\cdot b(\omega)}{c(\omega)} \\ &= \frac{1}{c(\omega)}\left([a(\omega)(1+R(\omega)) - b(\omega)S(\omega)] + j\,[b(\omega)(1+R(\omega)) + a(\omega)S(\omega)]\right), \end{aligned}$$

where

$$\begin{aligned} R(\omega) &= \mathrm{Re}\,(\Delta_2(j\omega))\,, \\ S(\omega) &= \mathrm{Im}\,(\Delta_2(j\omega)) \end{aligned}$$

and therefore $||\Delta_2(s)||_\infty < r_2 \equiv R(\omega)^2 + S(\omega)^2 < r_2^2 \;\forall\; \omega$. Hence, the largest value of r_2 for which the M-circle exclusion region (with centre x_c and radius r) for an uncertain system

$L_1(s)$ is avoided can be expressed as the first order formula:

$$\forall\, \omega,\, S(\omega),\, R(\omega):\; S(\omega)^2+R(\omega)^2<r_2^2 \implies \tag{3.39}$$
$$(a(\omega)(1+R(\omega))-b(\omega)S(\omega)-x_c c(\omega))^2+(b(\omega)(1+R(\omega))+a(\omega)S(\omega))^2-r^2c(\omega)^2>0.$$

By giving (3.39) as an input to the quantifier elimination algorithm, we obtain the largest value of r_2 (since it is the only non-quantified variable) for which (3.39) holds. An example of an application of the approach discussed here is given in Section 3.3.4.1.

3.3.3 Expression of SSV μ computation problem as a quantifier elimination problem in MIMO case

Consider the feedback interconnection depicted in Figure 3.6 with nominal system G being closed-loop stable and uncertainty structure $\Delta(\delta)$ given by (3.26) and repeated here for clarity:

$$\Delta(\delta)=\{\Delta:\Delta=\text{block diag}(\Delta_1(s),\ldots,\Delta_n(s)),\Delta_i\in H_\infty^{r_i\times r_i},||\Delta_i||_\infty\leq\delta\}. \tag{3.40}$$

Then the closed-loop system G is stable $\forall\Delta\in\Delta(\delta)$ if and only if:

$$\forall\Delta\in\Delta(\delta),\,\forall\omega,\det(I-G(j\omega)\Delta(j\omega))\neq0. \tag{3.41}$$

Introduce the smallest value of the uncertainty magnitude δ such that $\exists\;\Delta\in\Delta(\delta)$ which destabilises the plant:

$$\delta_{\min}(G(j\omega))=\min\{\delta:\det(I-G(j\omega)\Delta(j\omega))=0\text{ for some }\Delta\in\Delta(\delta)\}. \tag{3.42}$$

This optimisation problem can be expressed as a quantifier elimination problem

$$\exists\,S(\omega)_1,R(\omega)_1,\ldots,S(\omega)_n,R(\omega)_n:(z\geq\delta)\wedge(\det(I-G(j\omega)\Delta(j\omega))=0)\wedge(S(\omega)_i^2+R(\omega)_i^2\leq\delta^2), \tag{3.43}$$

which, by eliminating δ, can be written as:

$$\exists\,S(\omega)_1,R(\omega)_1,\ldots,S(\omega)_n,R(\omega)_n:(\det(I-G(j\omega)\Delta(j\omega))=0)\wedge(S(\omega)_i^2+R(\omega)_i^2\leq z^2). \tag{3.44}$$

Since expression (3.44) contains the optimisation variable z and frequency ω as unquantified variables, its output can be expressed as z function of ω (i.e. $z(\omega)$). Since the structured singular value μ is defined as

$$\mu(G(j\omega)) = \delta_{\min}(G(j\omega))^{-1} = \frac{1}{z(\omega)}, \tag{3.45}$$

this implies that expression (3.44) allows us to obtain the structured singular value μ. The uncertain plant is stable $\forall\Delta \in \Delta(\delta)$ with the uncertainty radius δ if and only if:

$$\forall\, \omega, \mu(G(j\omega)) < \delta^{-1}. \tag{3.46}$$

Hence, the actual exact expression for the structured singular value μ (expressed as an algebraic number) can be found by adding existential quantification of frequency ω to (3.44):

$$\exists\, \omega, S(\omega)_1, R(\omega)_1, \ldots, S(\omega)_n, R(\omega)_n : (\det(I - G(j\omega)\Delta(j\omega)) = 0) \wedge (S(\omega)_i^2 + R(\omega)_i^2 \leq z^2). \tag{3.47}$$

Because of the presence of multiple uncertainty bound definitions $S(\omega)_i^2 + R(\omega)_i^2 \leq z^2$ in (3.44) and (3.47), in general, a CAD-based quantifier elimination algorithm (described in Section 2.3.3) will be required to eliminate quantified variables from these first order formulas.

Now consider the uncertainty structure $\Delta_R(\delta)$ (3.30) containing only real parametric uncertainties and repeated here for clarity:

$$\Delta_R(\delta) = \{\Delta : \Delta = \operatorname{diag}(\Delta_1(s), \ldots, \Delta_n(s)), \Delta_i \in \mathbb{R}, |\Delta_i| \leq \delta\}. \tag{3.48}$$

For this kind of uncertainty structure, the quantifier elimination problem (3.44) becomes:

$$\exists\, \Delta_1, \Delta_2, \ldots, \Delta_n : \; (\det(I - G(j\omega)\Delta) = 0) \wedge (|\Delta_i| \leq \delta). \tag{3.49}$$

Similarly to (3.47), the actual value of the real structured singular value μ (expressed as an algebraic number) can be calculated by eliminating quantified variables from the expression:

$$\exists\, \omega,\; \Delta_1, \Delta_2, \ldots, \Delta_n : \; (\det(I - G(j\omega)\Delta) = 0) \wedge (|\Delta_i| \leq \delta). \tag{3.50}$$

Moreover, recall that a matrix is singular if and only if its rows (columns) are linearly dependent. Without loss of generality, let $a_i(\omega), i = 1, \ldots, n$ be the i-th row of the matrix $(I -$

$G(j\omega)\Delta)$. Then the quantifier elimination problem (3.49) is equivalent to

$$\exists\, \Delta_1, \Delta_2, \ldots, \Delta_n, k_1, k_2, \ldots, k_n :\ (k_1 \neq 0 \vee \ldots \vee k_n \neq 0) \wedge \sum_{i=1}^{n} k_i a_i(\omega) = 0_{(1\,\mathrm{x}\,n)} \wedge |\Delta_i| \leq \delta, \tag{3.51}$$

where $0_{(1\,\mathrm{x}\,n)}$ is a 1-by-n zero vector. It is important to notice that all quantified variables in (3.51) appear linearly and therefore Weispfenning's quantifier elimination algorithm (described in Section 2.3.2) is capable of eliminating all of them.

Additionally, in the case of the real uncertainty structure (3.48), the reciprocal of the structured singular value μ ($z = 1/\mu$) can be calculated by eliminating quantified uncertainties from the formula based on the Routh-Hurwitz stability criterion (Routh, 1877)

$$\forall\, \Delta_1, \Delta_2, \ldots, \Delta_n, (|\Delta_1| \leq \delta \wedge \ldots \wedge (|\Delta_n| \leq \delta) \implies (\text{Routh-Hurwitz conditions for } (I + L(\Delta_R, s))), \tag{3.52}$$

where $L(\Delta_R, s)$ is the open-loop transfer function. An example of such a computation will be given for a SISO system in Section 3.3.4.1.

The main advantage of computing structured singular value μ by eliminating quantified variables from (3.47) compared to using Proposition 2 on page 63 is that quantifier elimination based approach computes the actual value of μ rather than it's upper bound. Additionally, the value obtained by using quantifier elimination will be expressed as an algebraic number. Moreover, most implementations for computing upper bound of μ that use Proposition 2 do so on a finite frequency grid. Hence, such implementation may miss a critical frequency at which the maximal value of $\mu(G(j\omega))$ is obtained. This is clearly not the case for quantifier elimination based methods because of their algebraic nature.

In principle, given enough time, all the quantified variables in (3.47) can be eliminated by using CAD-based quantifier elimination procedure. Unfortunately, in practice, it turns out that this problem becomes computationally intractable even in the case of two norm-bounded uncertainties (i.e., $i = 2$) and a system with a single input and a single output. Hence, the main disadvantage of our approach is that because of the high computational complexity of the quantifier elimination problem, only very simple problems with a general norm-bounded uncertainty structure (3.40) can be considered.

Now consider computing the structured singular value μ for the system containing only real parametric uncertainties (3.48). Doing so by using a standard iterative branch and bound method (such as the one given in de Gaston and Safonov (1988) and discussed in Section 3.3.1) allows one to bound the value of μ to an arbitrary accuracy. In contrast, quantifier elimination based approach gives the exact value of μ (rather than a bound) in a form of an algebraic number. Moreover, in this case, Weispfenning's virtual substitution algorithm can be used since all the quantified variables in the structured singular value μ computation problem (3.51) appear linearly. Weispfenning's algorithm has better worst-case running time than CAD-based algorithm because it does not depend on the number of free variables. As a result, our approach is capable of computing real-μ for more complicated systems than it is for systems with a general norm-bounded uncertainty (3.40). Examples of such computation are presented in Sections 3.3.4.2 and 3.3.4.3 for systems with three parametric uncertainties and for the missile system with four parametric uncertainties, respectively.

3.3.4 Computational examples

3.3.4.1 Robustness of a simple SISO system

As described in Section 3.3.2 on page 65, consider an uncertain system of the form

$$L(s) = (1+\Delta_1(s))\,(1+\Delta_2(s))\,L_0(s) = (1+\Delta_1(s))\,L_1(s) \tag{3.53}$$

with $||\Delta_1(s)||_\infty < r_1 = 1/M$, $||\Delta_2(s)||_\infty < r_2, R(\omega) = \mathrm{Re}(\Delta_2(j\omega)), S(\omega) = \mathrm{Im}(\Delta_2(j\omega))$ and a nominal system (taken from Dorobantu et al. (2014))

$$L_0(s) = \frac{18.75s+225}{s^2+7.22s+246.5} \tag{3.54}$$

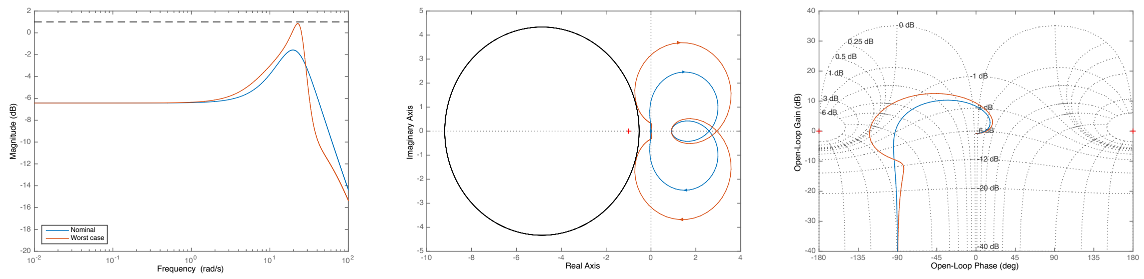
which is representative of the short period longitudinal dynamics of a small fixed wing aircraft.

Suppose we want to find the largest value of r_2 such that for all uncertainties $||\Delta_2(s)||_\infty < r_2$, the uncertain system $L_1(s) = (1+\Delta_2(s))L_0(s)$ does not enter the M-circle exclusion region of value $M = 10^{0.05} \approx 1.122$ (with centre $x_c = -4.86$, $y_c = 0$ and radius $r = 4.33$). This in turn guarantees that $\forall\, ||\Delta_2(s)||_\infty < r_2$:

- the magnitude of the complementary sensitivity $T_1(s)$ of the uncertain system $L_1(s)$ will stay below $20\log_{10}M = 1$ dB.
- $\mu_\Delta(L(s)) < 1$.

As was shown in Section 3.3.2, the calculation of this largest value of r_2 can be expressed as a quantifier elimination problem

$$\forall\, \omega,\ S(\omega),\ R(\omega):\ S(\omega)^2 + R(\omega)^2 < r_2^2 \implies p(\omega, S(\omega), R(\omega)) > 0, \tag{3.55}$$



(a) Magnitude of the complementary sensitivity function.

(b) Nyquist plot with the M-circle exclusion region.

(c) Nichols plot.

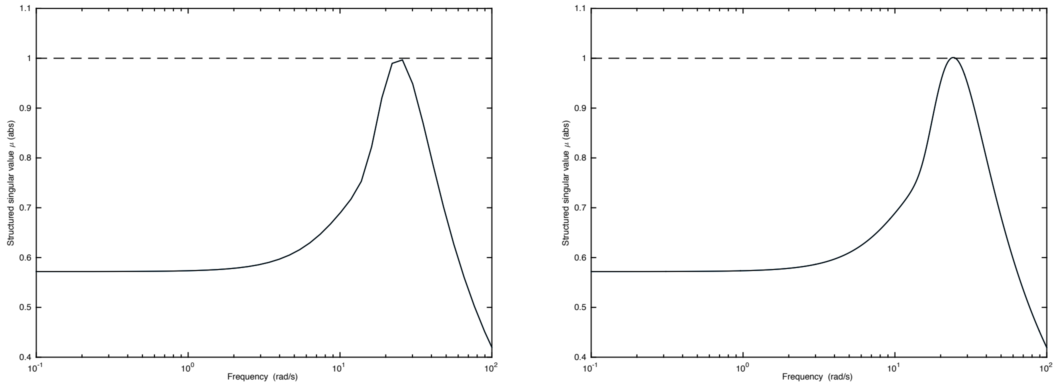
Fig. 3.7 Plots for nominal plant $L_0(s)$ (blue) and worst case uncertain plant $L_1(s)$ (red).

where $p(\omega, S(\omega), R(\omega))$ is a polynomial of the form:

$$p(\omega, S(\omega), R(\omega)) \equiv \tag{3.56}$$
$$(a(\omega)(1+R(\omega)) - b(\omega)S(\omega) - x_c c(\omega))^2 + (b(\omega)(1+R(\omega)) + a(\omega)S(\omega))^2 - r^2 c(\omega)^2.$$

By eliminating quantified variables from (3.55), the CAD-based quantifier elimination algorithm finds that $-0.547174 \leq r_2 \leq 0.547174$. Corresponding plots with $r_2 = 0.547174$ obtained with *MATLAB* Robust Control Toolbox (The MathWorks Inc., 2012a) are shown in Figure 3.7. It is worth noting that functions implemented in the Robust Control Toolbox (such as `mussv` for computing lower and upper bounds on the structured singular value μ) perform computations on an automatically determined finite frequency grid. As a consequence, Robust Control Toolbox claims that the system (3.53) with $||\Delta_1(s)||_\infty < \frac{1}{M} = 10^{-0.05}$ is robustly stable for $r_2 \leq 0.55$, as can be seen in Figure 3.8 (a). Increasing the density of the frequency grid at which bounds on the structured singular value μ are calculated by the Robust Control Toolbox results in the plot of μ depicted in Figure 3.8 (b), which clearly shows that the system is not robustly stable with $r_2 = 0.55$. Hence, this represents an advantage of computing the structured singular value μ via quantifier elimination methods. Since quantifier elimination procedures are algebraic, they are guaranteed not to miss the critical frequency ω or parameter combination at which the desired verification criterion is violated, unlike standard verification methods (such as the implementation of the algorithm in the Robust Control Toolbox for finding bounds on μ).

With $r_1 = 1/M = 10^{-0.05}$ and, given the level of uncertainty r_2, for which values of $R(\omega) = \mathrm{Re}(\Delta_2(j\omega))$ and $S(\omega) = \mathrm{Im}(\Delta_2(j\omega))$ is the system (3.53) robustly stable, i.e.



(a) Regular frequency grid. (b) Denser frequency grid.

Fig. 3.8 μ over frequency plots obtained via Robust Control Toolbox with $r_1 = 10^{-0.05}$, $r_2 = 0.55$.

$\mu_\Delta(L(s)) < 1$? This problem can be expressed as a quantifier elimination one:

$$\forall \omega : (S(\omega)^2 + R(\omega)^2 < r_2^2) \implies p(\omega,\, S(\omega),\, R(\omega)) > 0. \tag{3.57}$$

Semialgebraic regions in terms of $S(\omega)$ and $R(\omega)$ for which $\mu_\Delta(L(s)) < 1$ for various values of r_2 are given in Figure 3.9 (a). Moreover, by keeping r_2 as a variable in (3.57) (rather than assigning numerical values), a semialgebraic region describing the dependence of $S(\omega), R(\omega)$ on r_2 can be found, as depicted in Figure 3.9 (b). Additionally, consider the first order formula with quantified uncertainties and frequency ω as a free variable

$$\forall S(\omega), R(\omega) : (S(\omega)^2 + R(\omega)^2 < r_2^2) \implies p(\omega,\, S(\omega),\, R(\omega)) > 0, \tag{3.58}$$

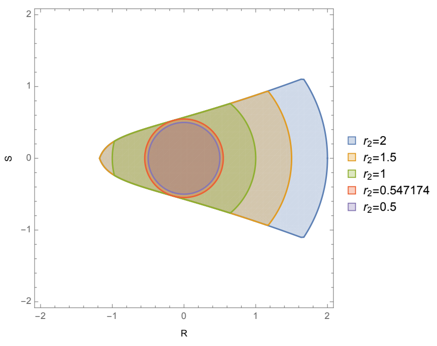
which could be used as an alternative to the quantifier elimination problem (3.55) for finding the largest value of r_2 for which the system (3.53) is robustly stable. Eliminating quantifiers from (3.58) gives us the expression of the form

$$r_2 = \sqrt{P_1(\omega) - \sqrt{P_2(\omega)}}, \tag{3.59}$$

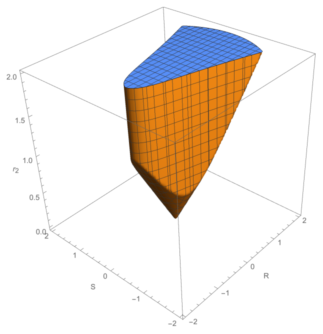
where $P_1(\omega)$ and $P_2(\omega)$ are rational functions of the frequency ω:

$$P_1(\omega) = \frac{0.120656\omega^4 - 54.673\omega^2 + 9009.44}{\omega^2 + 144}, \tag{3.60}$$

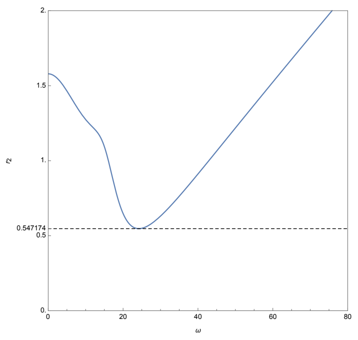
$$P_2(\omega) = \frac{0.0143667\omega^8 - 12.9837\omega^6 + 5036.16\omega^4 - 946985\omega^2 + 7.48275 \times 10^7}{(\omega^2 + 144)^2}. \tag{3.61}$$



(a) Semialgebraic regions in terms of $R(\omega)$, $S(\omega)$ for different values of r_2 for which the system $L(s)$ is robustly stable.



(b) Semialgebraic regions in terms of $R(\omega)$, $S(\omega)$ and r_2 for which the system $L(s)$ is robustly stable.



(c) r_2 dependence on frequency ω (3.59).

Fig. 3.9 Robustness computations for the SISO system of the form (3.53).

For the purpose of clear representation, coefficients of polynomials in (3.60) and (3.61) are approximations of the actual algebraic numbers that were part of the output of the quantifier elimination algorithm. The graph of the function (3.59) representing r_2 dependence on frequency ω is shown in Figure 3.9 (c). It can be seen that for $r_2 \leq 0.547174$, the system (3.53) is robustly stable, as expected.

Trying to calculate the structured singular value μ as a function of frequency, $\mu(L(j\omega))$, with complex uncertainties $||\Delta_1(s)||_\infty < \delta$, $||\Delta_2(s)||_\infty < \delta$ by eliminating quantifiers from (3.44) (Section 3.3.3, page 67) fails because of the high computational burden experienced by the CAD-based quantifier elimination algorithm.

Now, suppose that the uncertainties Δ_1 and Δ_2 are real and $|\Delta_1| \leq 1$, $|\Delta_2| \leq 1$. Then, by eliminating quantifiers from the formula (3.49) using Weispfenning's virtual substitution algorithm, we obtain the following dependence:

$$\delta(\omega) = \frac{1}{\mu(\omega)} \equiv \left(\omega = -\frac{\sqrt{\frac{7993}{2}}}{5} \wedge z \geq \frac{7\sqrt{\frac{53}{3}}}{25}\right) \vee \left(\omega = 0 \wedge z \geq \frac{\sqrt{\frac{943}{2}}}{15}\right) \vee \left(\omega = \frac{\sqrt{\frac{7993}{2}}}{5} \wedge z \geq \frac{7\sqrt{\frac{53}{3}}}{25}\right). \tag{3.62}$$

Therefore, the structured singular value μ for this system is:

$$\mu = \frac{25}{7\sqrt{\frac{53}{3}}} \approx 0.85 \leq 1. \tag{3.63}$$

Hence, the uncertain system (3.53) with real uncertainties $|\Delta_1| \leq 1$, $|\Delta_2| \leq 1$ is robustly stable.

Additionally, trying to find the reciprocal z of the structured singular value μ via the

Routh-Hurwitz based quantifier elimination problem (as described by (3.52)) results in the first order formula:

$$\forall\,\Delta_1,\Delta_2(|\Delta_1| \le z \wedge (|\Delta_2| \le z) \implies \left(\frac{1875}{100}(\Delta_1\Delta_2 + \Delta_1 + \Delta_2) + \frac{1875}{100} + \frac{722}{100} > 0\right) \wedge \left(225(\Delta_1\Delta_2 + \Delta_1 + \Delta_2) + \frac{2465}{10} + 225 > 0\right). \tag{3.64}$$

Eliminating quantifiers from (3.64) produces the result that is in agreement with (3.63), as expected.

As was noted in Section 3.3.3, in practice, quantifier elimination based approach can be used to compute structured singular value μ only for very simple systems with a general norm-bounded uncertainty structure (3.40). Hence, this motivates to concentrate our attention to systems with only real parametric uncertainties (3.48). These types of systems should be more amenable to our approach since, unlike in the norm-bounded uncertainty case, Weispfenning's virtual substitution algorithm can be used instead of CAD-based algorithm to eliminate quantifiers from (3.51). Sections 3.3.4.2 and 3.3.4.3 will illustrate this approach by considering systems with real parametric uncertainties.

3.3.4.2 Systems with three real parametric uncertainties

Consider a SISO system taken from de Gaston and Safonov (1988) with the open loop transfer function

$$L(s) = \frac{800(s+2)(1+\Delta_1)}{s(s+10)(s+4+\Delta_2)(s+6+\Delta_3)} \tag{3.65}$$

and three real parametric uncertainties $|\Delta_1| \le 0.1$, $|\Delta_2| \le 0.2$ and $|\Delta_3| \le 0.3$. Here Δ_1 represents multiplicative gain uncertainty while Δ_2 and Δ_3 are additive perturbations representing uncertainties in its pole position. This SISO system can be transformed into the feedback interconnection of the form shown in Figure 3.6 with

$$G(s) = \begin{bmatrix} \frac{-800(s+2)}{n(s)} & \frac{800(s+2)}{n(s)} & \frac{800(s+2)(s+4)}{n(s)} \\ \frac{s(s+10)(s+6)}{n(s)} & \frac{-s(s+10)(s+6)}{n(s)} & \frac{800(s+2)}{n(s)} \\ \frac{s(s+10)}{n(s)} & \frac{-s(s+10)}{n(s)} & \frac{-s(s+10)(s+4)}{n(s)} \end{bmatrix} \tag{3.66}$$

(here $n(s) = s^4 + 20s^3 + 124s^2 + 1040s + 1600$) and structured uncertainty Δ:

$$\Delta = \mathrm{diag}(\delta_1, \delta_2, \delta_3),\ \delta_1 = \frac{\Delta_1}{0.1}, |\delta_1| \le 1; \delta_2 = \frac{\Delta_2}{0.2}, |\delta_2| \le 1; \delta_3 = \frac{\Delta_3}{0.3}, |\delta_3| \le 1. \tag{3.67}$$

Computing the structured singular value μ for the system G (3.66) with uncertainty structure Δ (3.67) can (according to (3.51)) be expressed as a quantifier elimination problem of the form

$$\exists\ \delta_1, \delta_2, \delta_3, k_1, k_2, k_3 :\ (k_1 \neq 0 \vee k_2 \neq 0 \vee k_3 \neq 0) \wedge \sum_{i=1}^{3} k_i a_i = 0_{(1 \text{ x } n)} \wedge |\delta_i| \leq \delta, \qquad (3.68)$$

where $a_i, i = 1,2,3$ is the i-th row of the matrix $(I - G(j\omega)\Delta)$ and $0_{(1 \text{ x } n)}$ is a 1-by-n zero vector. Then, by eliminating quantifiers from (3.68) using Weispfenning's quantifier elimination algorithm, we obtain the $\frac{1}{\delta(\omega)} = \mu(\omega)$ expression in algebraic form

$$\mu(\omega) = \begin{cases} \frac{1}{10} & \text{for } \omega = 0, \\ \dfrac{1}{\frac{80(2\omega^2+15)}{3(\omega^2+20)} - \frac{5}{3}\sqrt{2}\sqrt{\frac{3\omega^6+320\omega^4+1200\omega^2}{(\omega^2+20)^2}}} & \text{for } 0 < \omega \leq \omega_1, \\ \dfrac{1}{\frac{20(9\omega^4+3100\omega^2+12000)}{3(3\omega^4+1100\omega^2+16000)} - \frac{10}{3}\sqrt{-\frac{9\omega^{10}+4200\omega^8+205200\omega^6-58720000\omega^4-979200000\omega^2-5184000000}{(3\omega^4+1100\omega^2+16000)^2}}} & \text{for } \omega_1 < \omega \leq \omega_2, \\ \dfrac{1}{\frac{-\omega^6-150\omega^4-2000\omega^2-20000}{5(\omega^2+20)^2} + \frac{1}{40}\sqrt{\frac{\omega^{14}+304\omega^{12}+31200\omega^{10}+1120000\omega^8+4000000\omega^6}{(\omega^2+20)^4}}} & \text{for } \omega > \omega_2, \end{cases} \qquad (3.69)$$

where

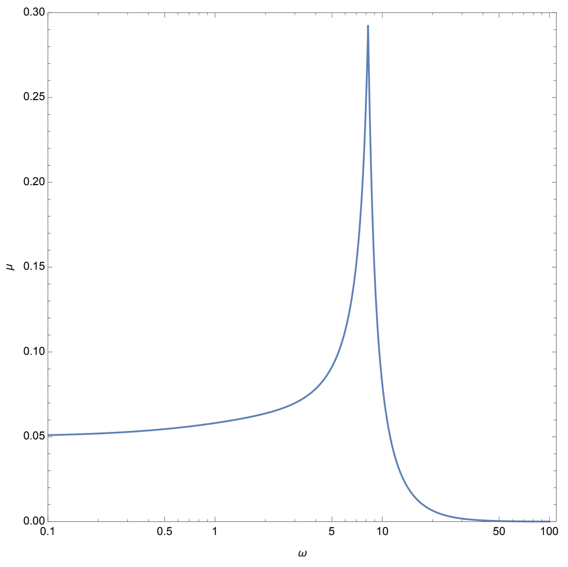
$$\omega_1 = \text{Root}\left[x^{10} + 520x^8 + 54800x^6 - 4710400x^4 - 106496000x^2 - 552960000, 2\right] \approx 8.2282,$$
$$\omega_2 = \text{Root}\left[9x^{10} + 4200x^8 + 365200x^6 - 42080000x^4 - 915200000x^2 - 5184000000, 2\right] \approx 8.6772.$$

Here Root$[f(x), k]$ represents the exact k-th root of the polynomial equation $f(x) = 0$ (all real roots come before the complex ones and are ordered in an increasing order; complex roots are ordered with respect to their real part first and then with respect to their negative imaginary part).

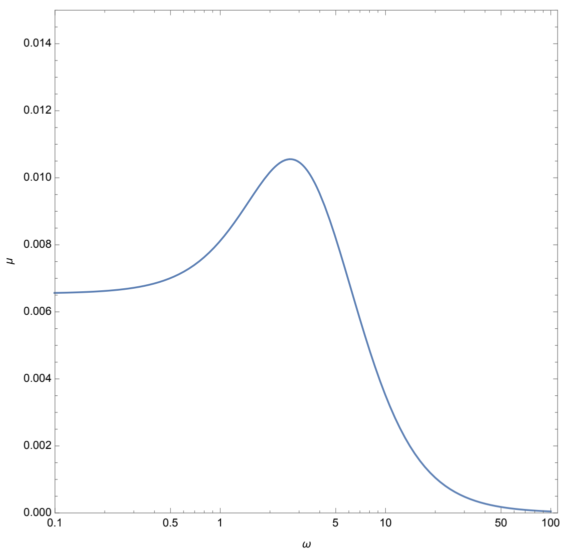
Moreover, consider the quantifier elimination problem (3.68) with additional existential quantification over frequency ω added, i.e.:

$$\exists\ \omega, \delta_1, \delta_2, \delta_3, k_1, k_2, k_3 :\ (k_1 \neq 0 \vee k_2 \neq 0 \vee k_3 \neq 0) \wedge \sum_{i=1}^{3} k_i a_i = 0_{(1 \text{ x } n)} \wedge |\delta_i| \leq \delta. \qquad (3.70)$$

Then, by firstly eliminating variables δ_1, δ_2, δ_3, k_1, k_2 and k_3 using Weispfenning's quantifier elimination algorithm (as was done to obtain (3.69)), and then eliminating frequency ω using CAD-based quantifier elimination algorithm, we compute the algebraic expression



(a) $\mu(\omega)$ graph for the system (3.66) subject to three real parametric uncertainties with uncertainty structure (3.67).



(b) $\mu(\omega)$ graph for the system (3.74) subject to repeated real parametric uncertainties with uncertainty structure (3.75) for $\omega > 0$.

Fig. 3.10 Graphs of $\mu(\omega)$ obtained using Weispfenning's quantifier elimination algorithm for systems subject to parametric uncertainties.

representing the exact value of the structured singular value μ:

$$\mu = \frac{\text{Root}\left[x^5 + 1260x^4 + 154800x^3 - 184420800x^2 + 10679040000x - 161740800000, 1\right]}{\text{Root}\left[x^5 - 120400x^4 + 5129120000x^3 - 8087769600000x^2 - 2936678400000000x + 7706050560000000000, 1\right]} \approx 0.292621. \tag{3.71}$$

Note that attempting to eliminate all quantified variables from (3.70) using CAD-based quantifier elimination algorithm does not produce an answer in 24 hours. The graph of the function $\mu(\omega)$ (3.69) is given in Figure 3.10 (a). System (3.65) was used in de Gaston and Safonov (1988) to illustrate their iterative branch and bound procedure for computing $\frac{1}{\mu}$ to an arbitrary accuracy for a system with real parametric uncertainties. After three iterations, the algorithm presented there arrives at the following upper and lower bounds for $\frac{1}{\mu(\omega)}$ for $\omega = 9$:

$$6.7256 \leq \frac{1}{\mu(9)} \leq 6.8976. \tag{3.72}$$

Note that by using (3.69), we can calculate the *exact* value of $\frac{1}{\mu(9)}$:

$$\frac{1}{\mu(9)} = \frac{131949\sqrt{15385} - 13580728}{408040} \approx 6.8271. \tag{3.73}$$

This illustrates the main difference between quantifier elimination based approach and branch and bound algorithms for computing structured singular value μ. The output of the quantifier elimination algorithm is an algebraic number representing the *exact* value of μ while the branch and bound based procedure is an iterative numerical algorithm providing tighter and tighter bounds on μ with each iteration.

Additionally, de Gaston and Safonov (1988) finds that the largest value of $\mu(\omega)$ for the system $G(s)$ (3.66) with the uncertainty structure (3.67) is $\mu = \frac{1}{3.44} \approx 0.2907$, obtained at the frequency $\omega = 8.22$ rad/s (since they have found that at $\omega = 8.22$, $\det(I - kG(j\omega)\Delta) \approx 0$ at one of the vertices of the hypercube D representing the uncertainty structure with $k = 3.44$). This is obviously not the case since, as we have found, the largest value of $\mu(\omega)$ is 0.292621 (3.71) and is obtained at $\omega = \omega_1 \approx 8.2282$. This is the case because the proposed quantifier elimination based procedure allows us to obtain the *exact* expression of $\mu(\omega)$ while the algorithm described in de Gaston and Safonov (1988) uses frequency gridding to derive bounds on $\mu(\omega)$ at each discrete frequency point, and an extensive frequency search is needed for this method in order to increase the accuracy. Hence, another advantage of the quantifier elimination based method when compared to this particular branch and bound method is that it is guaranteed not to miss the critical frequency at which the maximal value of $\mu(\omega)$ is obtained.

Now consider a nominal system $G(s)$ in Figure 3.6 taken from Chang et al. (1991) with a state-space representation (A, B, C)

$$A = \begin{bmatrix} -\frac{27}{10} & -2 & -\frac{3}{2} & -\frac{1}{2} \\ -\frac{3}{2} & -4 & -\frac{3}{2} & -\frac{3}{2} \\ -\frac{1}{5} & 0 & -3 & 0 \\ \frac{3}{2} & 2 & \frac{7}{2} & -\frac{7}{10} \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad C = \begin{bmatrix} -\frac{3}{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{3}{10} \\ -\frac{3}{10} & 0 & 0 & 0 \end{bmatrix} \tag{3.74}$$

and uncertainty structure with a repeated parametric uncertainty δ_2:

$$\Delta = \operatorname{diag}(\delta_1, \delta_2, \delta_2), \ |\delta_1| \le 1, |\delta_2| \le 1. \tag{3.75}$$

As before, computation of the structured singular value μ for the system $G = C(sI - A)^{-1}B$ (3.74) with the uncertainty structure (3.75) can be expressed as a quantifier elimination problem of the form analogous to (3.68). Hence, by eliminating the quantified variables, we obtain the $\frac{1}{\delta(\omega)} = \mu(\omega)$ expression in algebraic form

$$\mu(\omega) = \begin{cases} \frac{27}{98} & \text{for } \omega = 0, \\ \frac{1}{P(\omega)} & \text{for } \omega > 0, \end{cases} \tag{3.76}$$

where:

$$\begin{aligned}P(\omega) = \mathrm{Root}[&x^3\left(81\omega^2+162\right)+x^2\left(-180\omega^4-5787\omega^2-26424\right)\\&+x\left(1740\omega^4+49908\omega^2+261216\right)\\&-2000\omega^6-44080\omega^4-323724\omega^2-774448,1].\end{aligned} \tag{3.77}$$

As before, by eliminating frequency ω using CAD-based quantifier elimination algorithm from the quantified formula

$$\exists\, \omega: \left(\omega = 0 \wedge \delta \geq \frac{98}{27}\right) \vee (\omega > 0 \wedge \delta \geq P(\omega)), \tag{3.78}$$

we obtain that $\mu = \frac{1}{\delta} = \frac{27}{98} \approx 0.2755$. Note that

$$\lim_{\omega \to 0} \frac{1}{P(\omega)} = \frac{1}{\mathrm{Root}\left[81x^3-13212x^2+130608x-387224,1\right]} \approx 0.00654619, \tag{3.79}$$

i.e. $\mu(\omega)$ is not continuous at $\omega = 0$. The graph of the function $\mu(\omega)$ (3.76) for $\omega > 0$ is given in Figure 3.10 (b).

System (3.74) was used in Chang et al. (1991) to illustrate their iterative branch and bound procedure for computing structured singular value μ for systems subject to real parametric uncertainty. The algorithm presented there arrives at the result of $\mu = 0.2755$, which is in line with the result obtained by the quantifier elimination based procedure. Unlike the method for computing real μ developed in de Gaston and Safonov (1988), the algorithm presented in Chang et al. (1991) does not rely on frequency search and can accommodate systems subject to repeated real parametric uncertainties. Hence, the main difference between the quantifier elimination based approach proposed by us and the branch and bound procedure in Chang et al. (1991) is that quantifier elimination algorithm outputs the *exact* value of μ in a form of an algebraic number while procedure in Chang et al. (1991) is a numerical algorithm that, with each iteration, keeps reducing the gap between the lower and the upper bounds on μ until the desired accuracy is achieved.

3.3.4.3 System with four real parametric uncertainties

Consider a longitudinal rigid missile model taken from Reichert (1992). This nonlinear aerodynamic missile model has one control input (tail deflection), two outputs (acceleration and pitch rate) that are used by the autopilot and two states (angle of attack and pitch rate). The linearisation of this system is performed at a particular trim point, and, consequently,

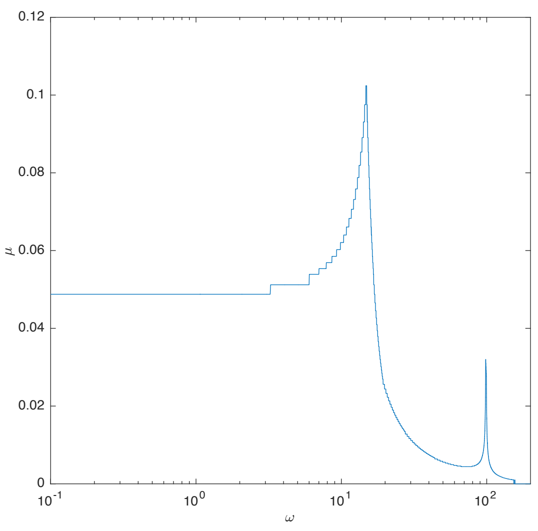


Fig. 3.11 Graph of μ value obtained using Weispfenning's quantifier elimination algorithm for a system subject to four real parametric uncertainties.

4 parametric uncertainties in the stability derivatives are directly introduced in the physical missile model.

By eliminating quantifiers from expression (3.51) with Weispfenning's quantifier elimination algorithm, we obtain the result illustrated in Figure 3.11. Attempting to perform the same calculation with a CAD algorithm does not produce an answer in 24 hours. The piecewise-constant nature of the graph in Figure 3.11 illustrates an important advantage of a quantifier-elimination-based verification. Since quantifier elimination is an algebraic procedure that analytically manipulates the expression (3.51), it is guaranteed not to miss frequencies at which a jump in the μ value occurs. This is clearly not the case for the structured singular value μ computation methods that rely on gridding.

3.4 Optimisation as a quantifier elimination problem

In this section, application of the quantifier elimination based approach to optimisation problems is considered. In Section 3.4.1, a general optimisation problem arising in the context of control synthesis and analysis problems and its conversion to an equivalent quantifier elimination problem is discussed. We note that, while in principle an algebraic solution to this general non-convex optimisation problem could be obtained by using the CAD-based quantifier elimination algorithm, it is infeasible in practice. Hence, in Section 3.4.2, we consider a particular (but common) type of optimisation problem arising in Model Predictive Control (MPC) that can be solved using Weispfenning's virtual term substitution algorithm and provide the procedure for obtaining this solution in Section 3.4.3. Then, in Section 3.4.4, we show how this algebraic solution to an optimisation problem, together with Weispfenning's algorithm, can be used to check recursive feasibility of the system, for both nominal and disturbed systems. Finally, Sections 3.4.5, 3.4.6 and 3.4.7 provide relevant application examples. Section 3.4.5 illustrates how the procedure developed in Section 3.4.3 can be used to obtain an explicit MPC solution for a linear system with a quadratic cost and linear constraints while Section 3.4.6 shows how the same method can be used to obtain an explicit MPC solution for an input-affine nonlinear system with a quadratic cost and linear constraints. Finally, in Section 3.4.7, we illustrate through several examples how Weispfenning's algorithm can be used to verify recursive feasibility of the MPC system.

3.4.1 Expression of a general optimisation problem as a quantifier elimination problem

Consider a general optimisation problem arising in the context of control synthesis and analysis questions

$$\begin{aligned} \min_{u_1,\ldots,u_m} \; & h(u_1,\ldots,u_m,x_1,\ldots,x_n) \\ \text{s.t. } & g_1(u_1,\ldots,u_m,x_1,\ldots,x_n) \leq 0, \\ & g_2(u_1,\ldots,u_m,x_1,\ldots,x_n) \leq 0, \\ & \ldots \\ & g_k(u_1,\ldots,u_m,x_1,\ldots,x_n) \leq 0, \end{aligned} \tag{3.80}$$

where $x_1,\ldots,x_n \in \mathbb{R}^i$ are states, $u_1,\ldots,u_m \in \mathbb{R}^j$ are inputs and

$$h, g_1, \ldots, g_k \in \mathbb{R}[u_1,\ldots,u_m,x_1,\ldots,x_n]$$

are polynomial functions of $u_1,\ldots,u_m,x_1,\ldots,x_n$. Then, the optimal value function $J(x_1,\ldots,x_n)$ can be obtained by eliminating quantified inputs from the definition of the the optimal value function:

$$\begin{aligned} & J(x_1,\ldots,x_n) \equiv \\ & \exists\, u_1,\ldots,u_m\, (J = h(u_1,\ldots,u_m,x_1,\ldots,x_n) \wedge g_1(u_1,\ldots,u_m,x_1,\ldots,x_n) \leq 0 \wedge \ldots \wedge g_k(u_1,\ldots,u_m,x_1,\ldots,x_n) \leq 0) \\ & \wedge [\forall u_1^{'},\ldots,u_m^{'} (g_1(u_1^{'},\ldots,u_m^{'},x_1,\ldots,x_n) \leq 0 \wedge \ldots \wedge g_k(u_1^{'},\ldots,u_m^{'},x_1,\ldots,x_n) \leq 0) \implies \\ & J \leq h(u_1^{'},\ldots,u_m^{'},x_1,\ldots,x_n)]. \end{aligned} \tag{3.81}$$

In principle, for any general optimisation problem of the form (3.80), the optimal value function $J(x_1,\dots,x_n)$ can be obtained by eliminating quantified variables from (3.81) with a CAD-based quantifier elimination algorithm. At the same time, in practice, (3.81) is an unnecessarily complicated quantified formula which increases the number of variables in the problem description from $m+n$ in (3.80) in to $2m+n$ in (3.81). This results in an increased execution time while running CAD-based quantifier elimination algorithm since, as was mentioned in Section 2.3.1, the upper bound on the size of the corresponding cylindrical algebraic decomposition grows doubly exponentially with the number of variables in the first order formula. This can be avoided by observing that in order to find the optimal value function $J(x_1,\dots,x_n)$, it is enough to eliminate m quantified inputs $u_1,\dots,u_m$ from the expression representing the lower bound on the feasible objective region:

$$\exists\, u_1,\dots,u_m(z \geq h \wedge g_1 \leq 0 \wedge \dots \wedge g_k \leq 0). \tag{3.82}$$

Once (3.82) is fed to a quantifier elimination algorithm, the output of it will be a quantifier-free expression of the form $z \geq J(x_1,\dots,x_n)$. It is then trivial to extract the optimal value function $J(x_1,\dots,x_n)$ from such an expression. Hence, for computational reasons, we will always express our optimisation problem as a quantifier elimination problem of the form (3.82).

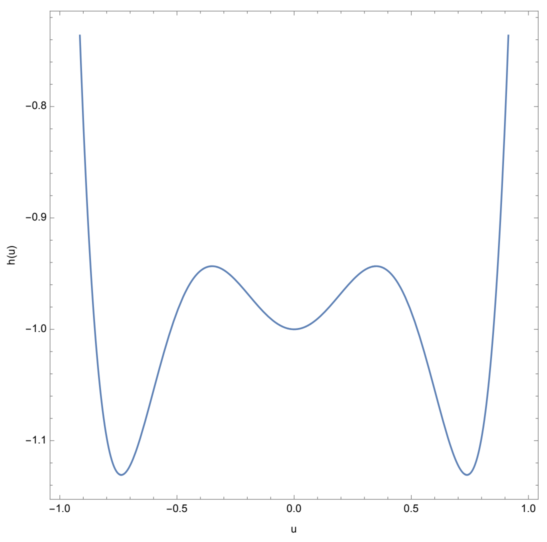
Similarly, optimal inputs $u_1^*,\dots,u_m^*$ for the general optimisation problem of the form (3.80) can be computed by eliminating quantified variables from the formula:

$$\begin{aligned}\forall u_1,\dots,u_m,\ (u_1 \neq u_1^* \vee \dots \vee u_m \neq u_m^*) \wedge (g_1(u_1,\dots,u_m) \leq 0 \wedge \dots \wedge g_k(u_1,\dots,u_m) \leq 0) \implies \\ h(u_1,\dots,u_m) \geq h(u_1^*,\dots,u_m^*) \wedge g_1(u_1^*,\dots,u_m^*) \leq 0 \wedge \dots \wedge g_k(u_1^*,\dots,u_m^*) \leq 0.\end{aligned} \tag{3.83}$$

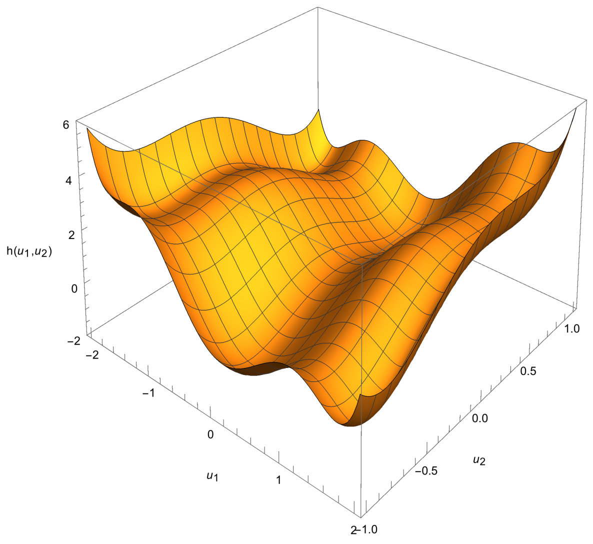
In principle, both (3.82) and (3.83) could be fed as an input to a general CAD-based quantifier elimination algorithm which would produce equivalent quantifier-free expressions corresponding to the optimal value function and optimal input, respectively. For example, consider a univariate nonconvex polynomial function $h(u)$ of the form

$$h(u) = 5u^6 - 5u^4 + u^2 - 1, \tag{3.84}$$

whose graph is presented in Figure 3.12 (a). The optimal value function J (which, in this case, is just a constant) of $h(u)$ can be computed by eliminating quantified variables from



(a) Graph of the univariate nonconvex polynomial function $h(u)$.



(b) Graph of the bivariate nonconvex polynomial six-hump camelback function $h(u_1,u_2)$.

Fig. 3.12 Graphs of example nonconvex univariate $(h(u))$ and bivariate $(h(u_1,u_2))$ polynomial objective functions.

either (3.81) or (3.82), with the result being:

$$J = \frac{4}{135}\left(-\sqrt{10}-35\right). \tag{3.85}$$

Computing (3.85) by eliminating quantifiers from (3.81) takes 0.09 seconds while doing so by eliminating quantifiers from (3.82) takes 0.004 seconds. Hence, as was noted before, in order to compute the optimal value function, it is more practical to use (3.82) as an input to the quantifier elimination algorithm. The optimal input u^* can be computed by eliminating quantified variables from the first order formula (3.83), with the result being

$$u^* = -\sqrt{\frac{1}{15}\left(\sqrt{10}+5\right)} \text{ or } u^* = \sqrt{\frac{1}{15}\left(\sqrt{10}+5\right)} \tag{3.86}$$

and the whole computation taking 0.01 seconds.

Now consider a bivariate nonconvex polynomial six-hump camelback function $h(u_1,u_2)$ (taken from Section 8.2.5 of A. Floudas et al. (1999))

$$h(u_1,u_2) = \left(\frac{u_1^4}{3}-\frac{21u_1^2}{10}+4\right)u_1^2+u_1u_2+u_2^2\left(4u_2^2-4\right), \tag{3.87}$$

whose graph is presented in Figure 3.12 (b). The optimal value function J of $h(u_1,u_2)$ is:

$$\begin{aligned} J = \mathrm{Root}[&1146617856000000000x^7 - 9581138804736000000x^6 + 27413621004828672000x^5 \\ &- 2368934311869598924 8x^4 - 22202693685841526784x^3 + 40987423246990311936x^2 \\ &- 7230107387480268288x - 3617973538199106125, 1] \approx -1.03163. \end{aligned} \tag{3.88}$$

Computing (3.88) by eliminating quantifiers from (3.81) takes 4.95 seconds while doing so by eliminating quantifiers from (3.82) takes 0.21 seconds. This again illustrates that computing the optimal value function J by eliminating quantified variables from the expression representing the lower bound on the feasible objective region (3.82) is more efficient in terms of execution time.

The optimal input (u_1^*,u_2^*) for $h(u_1,u_2)$ can be found via quantifier elimination to be

$$(u_1^*,u_2^*) = (r_1,r_2) \text{ or } (u_1^*,u_2^*) = (-r_1,-r_2),\ r_1,r_2 \in \mathbb{R}, \tag{3.89}$$

where

$$\begin{aligned} r_1 &= \mathrm{Root}[16000x^{14} - 201600x^{12} + 1038720x^{10} - 2798208x^8 + 4154880x^6 \\ &\qquad - 3227600x^4 + 1032400x^2 - 8125, 8] \approx 0.089842, \\ r_2 &= \mathrm{Root}[10485760x^{14} - 26214400x^{12} + 26214400x^{10} - 13279232x^8 + 3534848x^6 \\ &\qquad - 456704x^4 + 22144x^2 - 325, 3] \approx -0.712656. \end{aligned} \tag{3.90}$$

Computing the optimal input (u_1^*,u_2^*) (3.89) by eliminating quantified variables from (3.83) using CAD-based quantifier elimination algorithm took around 10 seconds. Notice that the analogous computation for the univariate case ($h(u)$ in (3.84)) took 0.01 seconds. This shows that, while it is possible in principle, attempting to compute the optimal input $u_1^*,\ldots,u_m^*$ for a general polynomial objective function $h(u_1,\ldots,u_m,x_1,\ldots,x_n)$ with three or more optimisation variables ($m \geq 3$) is an infeasible task in a practical sense.

Therefore, in Section 3.4.2, we will limit our attention to a particular set of optimisation problems arising in Model Predictive Control (MPC) that can be converted to quantifier elimination problems that Weispfenning's virtual term substitution algorithm (whose worst-case running time (in contrast to CAD) does not depend on the number of free variables n) could deal with. Afterwards, in Section 3.4.3, we will develop a procedure based on Weispfenning's quantifier elimination method for solving the MPC problem presented in Section 3.4.2.

3.4.2 Explicit MPC problem formulation

Consider a discrete-time linear time invariant system

$$x(i+1) = f(x(i), u(i)) = Ax(i) + Bu(i), \tag{3.91}$$

where $x \in \mathbb{R}^n$ is the state and $u \in \mathbb{R}^m$ is the input. Suppose we want to regulate (3.91) to the origin in such a way that polytopic constraints

$$\Psi(i) \equiv E(i)x(i) + F(i)u(i) \leq b(i) \tag{3.92}$$

are satisfied at all time steps $i = 0, 1, \ldots$. Assuming measurement of the current state $x(0)$ is available, one of the ways to solve this regulation problem by using MPC is to solve the following quadratic optimisation problem

$$\begin{aligned} \min_{u(0),\ldots,u(N-1)} \; & \sum_{i=0}^{N-1} \left(x(i)^T Q x(i) + u(i)^T R u(i) \right) + x(N)^T P x(N) \\ & \text{s.t. } \Psi(i),\; i = 0, \ldots, N-1,\; E(N)x(N) \leq b(N) \end{aligned} \tag{3.93}$$

where at each time step

$$x(i) = A^i x(0) + \sum_{j=0}^{i-1} A^{i-1-j} B u(j) \tag{3.94}$$

is the predicted state at time step i obtained given the current state $x(0)$ and the input sequence $\{u(0), u(1), \ldots, u(N-1)\}$, N is the control horizon and Q, R and P are state, input and terminal costs, respectively. In general, it is common practice to make the assumption that $Q = Q^T \geq 0$, $R = R^T > 0$, $P \geq 0$ (where $Q > 0$ and $Q \geq 0$ denote that the matrix Q is positive-definite and positive-semidefinite, respectively) in order to make the optimisation problem (3.93) convex.

Standard implementation of MPC relies on online optimisation to obtain the optimal input sequence $\{u^*(0), u^*(1), \ldots, u^*(N-1)\}$ at each time step by solving (3.93), an optimal control problem over a finite horizon. In the usual receding horizon approach, only the first input $u^*(0)$ of the optimal input sequence obtained is applied to the controlled system (3.91). At the following time step, the measurement of the new current state $x(0)$ is obtained and the whole procedure is repeated.

On the other hand, an analytic expression of the optimal input $u^*(0)$ can be obtained offline as a piecewise-affine function of the state

$$u_j^*(0) = K_j x(0) + G_j \text{ if } H_j x(0) \leq k_j,\ j = 1,\dots,M, \tag{3.95}$$

where $K_j \in \mathbb{R}^{m\times n}$, $G_j \in \mathbb{R}^m$ and $H_j x(0) \leq k_j$ $j = 1,\dots,M$ represent M polytopic regions in terms of the current state $x(0)$ where $u_j^*(0)$ is the optimal control input. This approach, called explicit MPC, is useful when implementation of MPC via online optimisation algorithm is not computationally feasible — for example, in cases of systems with high sampling rates. Explicit MPC allows to move most of the computation offline, while online computation gets reduced to a relatively simple computation of the optimal input (3.95) depending on the value of the current state $x(0)$.

The standard way of obtaining the explicit MPC solution (3.95), developed in Bemporad et al. (2002), is via multiparametric programming which treats the current state $x(0)$ as a parameter. In order to provide a summary of this algorithm, it is helpful to express the optimisation problem (3.93) in an appropriate equivalent form first. Let:

$$U = \left[u^T(0), u^T(1), \dots, u^T(N-1)\right]^T \in \mathbb{R}^{Nm}, \qquad X = \left[x^T(1), x^T(2), \dots, x^T(N)\right]^T \in \mathbb{R}^{Nn}. \tag{3.96}$$

Then (3.94) for $i = 0,\dots,N$ can be written as

$$X = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix} x(0) + \begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix} U = \Phi x(0) + \Gamma U, \tag{3.97}$$

where Φ and Γ are the so called prediction matrices. Also note that the cost function in (3.93) can be written as

$$\begin{aligned} V_N(x(0),U) &\equiv \sum_{i=0}^{N-1} \left(x(i)^T Q x(i) + u(i)^T R u(i)\right) + x(N)^T P x(N) \\ &= x(0)^T Q x(0) + X^T \Omega X + U^T \Psi U, \end{aligned} \tag{3.98}$$

where:

$$\Omega = \operatorname{diag}(Q, Q, \dots, Q, P) \in \mathbb{R}^{Nn} \times \mathbb{R}^{Nn}, \Omega \geq 0, \tag{3.99}$$

$$\Psi = \operatorname{diag}(R, R, \dots, R) \in \mathbb{R}^{Nm} \times \mathbb{R}^{Nm}, \Psi > 0. \tag{3.100}$$

Then, by taking the advantage of the expression (3.97), (3.98) can be written as

$$V_N(x(0),U) = \frac{1}{2}U^T HU + x(0)^T FU + x(0)^T Yx(0), \tag{3.101}$$

where:

$$H = 2(\Gamma^T \Omega \Gamma + \Psi),\ F = 2\Phi^T \Omega \Gamma,\ Y = Q + \Phi^T \Omega \Phi. \tag{3.102}$$

Note that since Ω and Ψ are diagonal matrices with $\Omega \geq 0$, $\Psi > 0$, $H = H^T > 0$. In a similar fashion, the set of polytopic constraints $\Psi(i), i = 0, \ldots, N-1$ (3.92), together with a terminal constraint $E(N)x(N) \leq b(N)$, can be expressed as:

$$\underbrace{\begin{bmatrix} E(0) \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{D_1} x(0) + \underbrace{\begin{bmatrix} 0 & \ldots & 0 \\ E(1) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & E(N) \end{bmatrix}}_{D_2} X + \underbrace{\begin{bmatrix} F(0) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & F(N-1) \\ 0 & \ldots & 0 \end{bmatrix}}_{D_3} U \leq \underbrace{\begin{bmatrix} b(0) \\ b(1) \\ \vdots \\ b(N) \end{bmatrix}}_{W}. \tag{3.103}$$

Then, by taking advantage of the expression (3.97), (3.103) can be written as

$$GU \leq W + Ex(0), \tag{3.104}$$

where

$$G = D_2\Gamma + D_3,\ E = -(D_1 + D_2\Phi). \tag{3.105}$$

Hence, this allows us to express optimisation problem (3.93) as

$$\begin{aligned} &\min_U \frac{1}{2}U^T HU + x(0)^T FU \\ &\text{s.t. } GU \leq W + Ex(0) \end{aligned} \tag{3.106}$$

which is a convex quadratic optimisation problem dependant on the current state $x(0)$. Note that the term $x(0)^T Yx(0)$ in (3.101) has been omitted in (3.106), as only the optimiser U is needed. Finally, according to Bemporad et al. (2002), by defining

$$z = U + H^{-1}F^T x(0), \tag{3.107}$$

the optimisation problem (3.106) can be written as an equivalent quadratic optimisation problem of the from

$$\begin{aligned}&\min_{z}\frac{1}{2}z^T Hz\\&\text{s.t. } Gz \leq W+Sx(0)\end{aligned} \tag{3.108}$$

where $S=E+GH^{-1}F^T$. Then, the first-order Karush-Kuhn-Tucker (KKT) optimality conditions (Bazaraa et al., 1993) for the convex quadratic optimisation problem (3.108) are

$$Hz+G^T\lambda=0,\ \lambda\in\mathbb{R}^q \tag{3.109}$$
$$\lambda_i(G^iz-W^i-S^ix(0))=0,\ i=1,\dots,q \tag{3.110}$$
$$\lambda_i\geq 0,\ i=1,\dots,q \tag{3.111}$$
$$Gz\leq W+Sx(0), \tag{3.112}$$

where $G^iz-W^i-S^ix(0)$ represents the i-th row of the matrix $Gz-W-Sx(0)$. Now, let $\overline{G}$, $\overline{W}$ and $\overline{S}$ correspond to some combination of active constraints (i.e., the set of constraints $\overline{G}z=\overline{W}+\overline{S}x(0)$ out of the constraints $Gz\leq W+Sx(0)$ in the optimisation problem (3.108)), with the assumption that the rows of $\overline{G}$ are linearly independent. Also, let $\overline{\lambda}$ denote the vector of Lagrange multipliers corresponding to these active constraints. Then, from (3.109) and $\overline{G}z=\overline{W}+\overline{S}x(0)$:

$$\overline{\lambda}=-(\overline{G}H^{-1}\overline{G}^T)^{-1}(\overline{W}+\overline{S}x(0)). \tag{3.113}$$

Note that $(\overline{G}H^{-1}\overline{G}^T)^{-1}$ exists in (3.113) because of the assumption made that the rows of $\overline{G}$ are linearly independent. Then, from (3.109) and (3.113), the solution to the optimisation problem (3.108) when the constraints represented by $\overline{G}$, $\overline{W}$ and $\overline{S}$ are active is:

$$z=-H^{-1}\overline{G}^T\overline{\lambda}=H^{-1}\overline{G}^T(\overline{G}H^{-1}\overline{G}^T)^{-1}(\overline{W}+\overline{S}x(0)). \tag{3.114}$$

Note that

$$U=z-H^{-1}F^Tx(0)=[u(0),u(1),\dots,u(N-1)]^T, \tag{3.115}$$

and therefore (3.114) represents the optimal solution to the optimisation problem (3.93) in some polyhedral state space region R. Region R is characterised by the corresponding dual feasibility (3.111) and primal feasibility (3.112) conditions, i.e.

$$\begin{aligned}R\equiv\{x(0)\in X\subseteq\mathbb{R}^n:&GH^{-1}\overline{G}^T(\overline{G}H^{-1}\overline{G}^T)^{-1}(\overline{W}+\overline{S}x(0))\leq W+Sx(0)\\&\wedge(\overline{G}H^{-1}\overline{G}^T)^{-1}(\overline{W}+\overline{S}x(0))\geq 0\},\end{aligned} \tag{3.116}$$

where $X \subseteq \mathbb{R}^n$ represents the set of states $x(0)$ for which the optimisation problem (3.108) is feasible. In other words, the region R (3.116) represents the largest set of states $x(0) \in \mathbb{R}^n$ such that the combination of the active constrains (represented by $\overline{G}$, $\overline{W}$ and $\overline{S}$) at the minimiser represented by (3.114) remains unchanged. The iterative procedure, developed in Bemporad et al. (2002), partitions the set $X \subseteq \mathbb{R}^n$ into mutually disjoint polyhedral regions of the form:

$$R_j = \left\{x(0) \in X : H_j x(0) \leq k_j\right\},\ j = 1, \ldots, M. \tag{3.117}$$

For each one of those regions, the associated optimal solution z to the optimisation problem (3.108) is found from (3.114). It is then trivial to obtain the corresponding optimal receding horizon control law in that region:

$$u_j^*(0) = [I_m\, 0\, \ldots\, 0]\, U = [I_m\, 0\, \ldots\, 0]\,(z - H^{-1}F^T x(0)),\ j = 1, \ldots, M. \tag{3.118}$$

In (3.118), $I_m \in \mathbb{R}^m \times \mathbb{R}^m$ represents an $m \times m$ identity matrix. The algorithm, developed in Bemporad et al. (2002), performs these computations to obtain the explicit MPC solution by first calling the procedure *partition(Y)* with $Y = X$ (where $X \subseteq \mathbb{R}^n$ is the set of states $x(0)$ for which the optimisation problem (3.108) remains feasible). The procedure *partition(Y)* proceeds as follows:

procedure partition(Y)

Step 1 Let $x_0 \in Y$ and ε be a solution to a particular linear optimisation problem and, if $\varepsilon \leq 0$, exit the partition procedure (for more details, see Bemporad et al. (2002)).
Step 2 For $x(0) = x_0$, compute the optimal solution z_0 to the quadratic optimisation problem (3.108).
Step 3 For $z = z_0$, $x(0) = x_0$, determine the set of active constraints and build the matrices $\overline{G}$, $\overline{W}$ and $\overline{S}$ representing those active constraints. If $r = \text{rank}\,\overline{G} < l$ where l is the number of rows of $\overline{G}$, then take a subset of r linearly independent rows, and redefine $\overline{G}$, $\overline{W}$ and $\overline{S}$ accordingly.
Step 4 Determine the optimal solution z from (3.114).
Step 5 Obtain the compact representation of the region R (3.116) (where the optimal solution z obtained in *step 4* holds) by removing redundant inequalities from R.
Step 6 Suppose that the polyhedral region $R \subseteq Y$, obtained in *step 5*, is of the form:

$$R \equiv \{x(0) \in Y : Ax(0) \leq b\}. \tag{3.119}$$

Then split the region $Y \setminus R$ into mutually disjoint regions $R_1, R_2, \ldots R_i$, i.e.

$$Y \setminus R \equiv R_1 \cup R_2 \cup \ldots \cup R_i,\ i = 1, \ldots, \dim(b),\ R_i \cap R_j = \emptyset\ \forall i \neq j, \tag{3.120}$$

where

$$R_i = \left\{x(0) \in Y : (A^i x(0) > b^i) \wedge (A^j x(0) \leq b^j\ \forall j < i)\right\},\ i = 1, \ldots, \dim(b), \tag{3.121}$$

with A^i representing the i-th row of the matrix A.

Step 7 For each one of the new regions $R_1, R_2, \ldots R_i$, $i = 1, \ldots, \dim(b)$, execute *partition* (R_i).

end procedure

Finally, for all the regions where the optimal solution z (as defined by (3.114)) is the same and whose union is the convex set, such a union is computed as described in Bemporad et al. (2001).

In the following section, we propose an alternative method for obtaining the optimal explicit MPC control law (3.95) for linear time invariant systems (3.91) with a quadratic objective and polytopic constraints (3.93) by using quantifier elimination.

3.4.3 Explicit MPC solution via quantifier elimination

Consider the problem of obtaining the optimal explicit MPC control law (3.95) for linear time invariant systems (3.91) with a quadratic objective and polytopic constraints (3.93). In all examples considered, the efficient Weispfenning's 'quantifier elimination by virtual substitution' algorithm is applicable, with all quantified variables appearing at most quadratically. The structure of the control problem considered makes it likely that the quantification pattern of the resulting quantifier elimination problem will be such that this holds more generally. At the end of this section, we will discuss which types of nonlinear systems the proposed procedure could be applied to as well, together with the main differences between the quantifier elimination based approach and the standard method (developed in Bemporad et al. (2002) and summarised in Section 3.4.2) for obtaining the optimal explicit MPC control law.

Let $L(i) = x(i)^T Q x(i) + u(i)^T R u(i)$ denote the i'th stage cost in the objective function in (3.93). Then, in a dynamic programming fashion, the single optimisation problem (3.93) in Nm variables $\{u(0), \ldots, u(N-1)\}$ can be expressed as N optimisation problems in m

variables:

$$\begin{aligned}
J_{N-1}(x(N-1)) = \min_{u(N-1)} h_{N-1}(u(N-1),x(N-1)) =& \\
\min_{u(N-1)} L(N-1) + x(N)^T P x(N)& \\
\text{s.t. } C(N-1) \equiv \{x(N) = Ax(N-1) + Bu(N-1) \wedge& \\
\Psi(N-1) \wedge E(N)x(N) \leq b(N)\}&
\end{aligned}$$

$$\begin{aligned}
J_{N-2}(x(N-2)) = \min_{u(N-2)} h_{N-2}(u(N-2),x(N-2)) =& \\
\min_{u(N-2)} L(N-2) + J(x(N-1))& \\
\text{s.t. } C(N-2) \equiv \{x(N-1) = Ax(N-2) + Bu(N-2) \wedge& \\
\Psi(N-2)\}& \\
\vdots&
\end{aligned}$$

$$\begin{aligned}
J_0(x(0)) = \min_{u(0)} h_0(u(0),x(0)) =& \\
\min_{u(0)} L(0) + J(x(1))& \\
\text{s.t. } C(0) \equiv \{x(1) = Ax(0) + Bu(0) \wedge \Psi(0)\}&
\end{aligned}$$

According to (3.82), each of these N optimisation problems can be expressed as a quantifier elimination problem

$$\exists u(i)\,(z \geq h_i(u(i),x(i)) \wedge C(i))\,,\; i = 0,\dots,N-1, \tag{3.122}$$

where z represents a slack variable which will be used to obtain the optimal value function $J_i(x(i))$. Applying the CAD-based quantifier elimination algorithm (see Section 2.3.3) to (3.122) would produce an equivalent quantifier-free formula with no input $u(i)$ dependence

$$\bigcup_{j=1,\dots,l_i} \big(x(i) \in F_j(i) \wedge z \geq J_j(x(i))\big) \tag{3.123}$$

with l_i disjoint polytopic regions ($F_l(i) \cap F_m(i) = \varnothing\ \forall l \neq m, l, m \in 1,\dots,l_i$) whose union is the set of feasible states:

$$F_1(i) \cup \ldots \cup F_{l_i}(i) \equiv \{x(i) : \exists u(i) \text{ such that } C(i)\}. \tag{3.124}$$

Additionally, each one of the regions $F_j(i), j = 1,\dots,l_i$ would have an associated optimal value function $J_j(x(i))$ from which we could obtain an optimal input $u^*(i)$. It is important to note that the CAD-based quantifier elimination algorithm would produce the output (3.123) that is "solved", i.e. regions $F_j(i)$ would be mutually disjoint. This is the case because the algorithm works by splitting the space $\mathbb{R}^m$ into disjoint cells in each of which all polynomials in the set $\{z \geq h_i(u(i), x(i)), C(i)\}$ are either positive, negative or zero. Unfortunately, as was discussed in Section 3.4.1, it is not practically feasible to use the CAD-based quantifier elimination algorithm because of the high execution time needed.

Hence, we apply Weispfenning's algorithm to (3.122). In the single input case, i.e. $m = 1$, then doing so produces an equivalent quantifier-free formula with no input $u(i)$ dependence

$$\bigcup_{j=1,\ldots,n_i} \big(x(i) \in O_j(i) \wedge z \geq p_j(x(i))\big) \tag{3.125}$$

with $p_j(x(i))$ a quadratic function of $x(i)$ and n_i overlapping regions $O_j(i)$ whose union is the set of feasible states:

$$O_1(i) \cup \ldots \cup O_{n_i}(i) \equiv \{x(i) : \exists u(i) \text{ such that } C(i)\}. \tag{3.126}$$

Applying Weispfenning's algorithm results in a set of overlapping regions $O_j(i), j \in 1, \ldots, n_i$ because, unlike the CAD-based quantifier elimination algorithm, Weispfenning's algorithm just eliminates the quantified variables but does not "solve" the resulting quantifier-free formula (see Section 2.3.2). Consequently, since the regions $O_j(i), j \in 1, \ldots, n_i$ overlap with each other, to obtain $J_i(x(i))$ we have to find the smallest $p_j(x(i))$ in each intersection of those regions. Therefore, region $O_1(i) \cup \ldots \cup O_{n_i}(i)$ is split into disjoint regions $D_1(i), \ldots, D_{m_i}(i)$ such that $D_l(i) \cap D_m(i) = \varnothing \; \forall l, m \in 1, \ldots, m_i, l \neq m$ according to the way overlapping regions intersect each other. This is done by picking two overlapping regions, $O_m(i), O_n(i), m, n \in 1, \ldots, n_i$, splitting them into three mutually disjoint regions $O_m(i) \cap O_n(i)$, $O_m(i) \setminus (O_m(i) \cap O_n(i))$ and $O_n(i) \setminus (O_m(i) \cap O_n(i))$, and repeating this procedure recursively until all the regions become mutually exclusive. Then for all $D_l(i), l \in 1, \ldots, m_i$, pick a point $x \in D_l(i)$ and evaluate all quadratic functions $p_j(x(i))$ corresponding to the regions $O_m(i), m \in 1, \ldots, n_i$ satisfying $O_m(i) \cap D_l(i) \neq \varnothing$ at $x(i) = x$. The quadratic function $p_j^*(x(i))$ with the smallest value $p_j(x)$ represents the candidate optimal value function in the region $D_l(i)$. If it is smallest everywhere in the region, as might be expected from the explicit MPC solution, it would follow that $p_j^*(x(i)) \equiv J_i(x(i)) \; \forall x(i) \in D_l(i)$, otherwise the boundary would be more complex. Finally, after connecting neighbouring regions with the same optimal value function, $J_i(x(i))$ is obtained (see Figure 3.13 (c) for an illustration).

In order to find the optimal explicit MPC control law $u^*(0)$, we have to obtain the expression for the optimal value function $J_0(x(0))$ first. To achieve this, apply the quantifier elimination algorithm to the first order formula (3.122) with $i = N-1$, then obtain $J_{N-1}(x(N-1))$ via steps described in the previous paragraph. Next apply the quantifier elimination algorithm to (3.122) with $i = N-2$, with the $J_{N-1}(x(N-1))$ expression just obtained substituted into the formula. Repeat until the expression for $J_0(x(0))$ is obtained.

Subsequently, the process to obtain the optimal input $u^*(0)$ is as follows. Suppose we are interested in the optimal input $u^*(0)$ in one of the disjoint regions $D_j(0), j \in 1, \ldots, m_i$. Firstly, the equation

$$h_0(x(0), u(0)) = J_0(x(0)) \tag{3.127}$$

is solved for $u(0)$. Since the equation (3.127) is quadratic in $u(0)$, it has two state-dependent solutions, $u_1(x(0))$ and $u_2(x(0))$. In order to find which one of these two solutions is the optimal input $u^*(0)$, let $C(0, u(x(0)))$ denote the input-independent constraint expression obtained once the input expression $u(x(0))$ is substituted in the constraint set $C(0)$. Consider the regions

$$R_1 \equiv C(0, u_1(x(0))) \wedge D_j(0), \tag{3.128}$$

$$R_2 \equiv C(0, u_2(x(0))) \wedge D_j(0). \tag{3.129}$$

One of those regions is empty while the other is not. Assume, without loss of generality, that the region R_1 is non-empty. This, in turn, means that $u_1(x(0))$ is the optimal input, i.e.:

$$u^*(0) = u_1(x(0)). \tag{3.130}$$

The region that is non-empty can be determined by feeding the quantified expressions

$$\exists\, x(0) R_1, \tag{3.131}$$

$$\exists\, x(0) R_2 \tag{3.132}$$

to Weispfenning's quantifier elimination algorithm. Since both expressions are fully quantified, the quantifier-free formula will be either $True$ or $False$. Again, suppose, without loss of generality, that Weispfenning algorithm outputs that (3.131) is equivalent to $True$ — this means that $u_1(x(0))$ is the optimal input, i.e. $u^*(0) = u_1(x(0))$. Although only $u^*(0)$ is applied to the system (3.91), notice that we could have obtained optimal $u^*(1), \ldots, u^*(N-1)$, by repeating the described procedure for equations analogous to (3.127). In Section 3.4.5, the approach described in this section is used to find the explicit MPC solution for a particular linear MPC system with a quadratic cost and linear constraints.

In general, Weispfenning's algorithm is applicable as long as the MPC problem can be expressed as an equivalent quantifier elimination one with particular quantification structure (i.e., all quantified variables appear at most quadratically). This suggests that, as long as this quantification requirement is met, the procedure discussed in this section to obtain explicit MPC solution via Weispfenning's algorithm can be used for nonlinear systems. This is indeed the case if we limit our attention to the same quadratic MPC problem (3.93), but with input-affine nonlinear system dynamics of the form

$$x(i+1) = g(x(i)) + Bu(i), \tag{3.133}$$

where $g(x(i))$ is a polynomial function of the state $x(i)$. In Section 3.4.6, the approach described in this section is used to find the explicit MPC solution for a system with dynamics of the form (3.133) together with a quadratic cost and linear constraints.

There are several differences to be noted between the quantifier elimination based approach discussed in this section and the standard method (developed in Bemporad et al. (2002) and summarised in Section 3.4.2) for obtaining the optimal explicit MPC control law (3.95). First of all, it was assumed in (3.93) that the state, input and terminal costs Q, R and P, respectively, satisfy the positive-definiteness and positive-semidefiniteness conditions:

$$Q = Q^T \geq 0,\ R = R^T > 0,\ P \geq 0. \tag{3.134}$$

This is a standard assumption in literature in order to make the optimisation problem (3.93) (ant its equivalent representations (3.106) and (3.108)) to be a convex optimisation problem. No such assumption has to be made for the quantifier elimination based method since, as was discussed and illustrated in Section 3.4.1, quantifier elimination is capable of dealing with a general nonconvex optimisation problem. Additionally, as long as the nonlinear MPC problem can be expressed as an equivalent quantifier elimination one with a particular quantification structure that Weispfenning's quantifier elimination algorithm can deal with, no additional work is needed to adapt the proposed method in order to make it applicable to nonlinear systems. Finally, although the command actions provided by the explicit feedback control law developed in Bemporad et al. (2002), by solving the optimisation problem (3.93) at each time step and by the explicit feedback control law obtained using quantifier elimination based approach all match, the quantifier elimination based approach is an algebraic procedure that expresses the explicit control law (3.95) using *exact* algebraic numbers.

3.4.4 Checking feasibility via quantifier elimination for a linear system

Once the explicit MPC solution for the system is obtained using the procedure based on the Weispfenning's quantifier elimination algorithm (as discussed in Section 3.4.3), it can be used to analyse feasibility of the optimisation problem (3.93).

Let F_0 be the set of initial states $x(0)$ for which the optimisation problem (3.93) is feasible (i.e., has a solution). Then, F_0 can be expressed as

$$F_0 \equiv \exists u(0) \exists u(1) \dots \exists u(N-1) \bigwedge_{i=0}^{N-1} \Psi(i) \wedge (E(N)x(N) \leq b(N)) \tag{3.135}$$

and therefore can be calculated by substituting expressions for predicted states $x(i) = A^i x(0) + \sum_{j=0}^{i-1} A^{i-1-j} Bu(j)$, $i = 1, \dots, N-1$, followed by eliminating quantifiers using Weispfenning's algorithm. Weispfenning's algorithm is applicable in this case since all quantified variables in (3.135) appear linearly. Additionally, let the set $I_0 \equiv \mathbb{R}^n \setminus F_0$ denote initial states $x(0)$ for which optimisation problem (3.93) is *not* feasible.

Similarly, let F_1 be the set of states x for which (3.93) remains feasible after a single control update, assuming that the optimal control law $u^*(0)$ (i.e., a piecewise control law obtained according to the procedure described in Section 3.4.3) is applied in the initial step. Then

$$F_1 \equiv \exists u \left(u_0^*(x) \wedge x \in F_0 \wedge Ax + Bu \in F_0 \right) \tag{3.136}$$

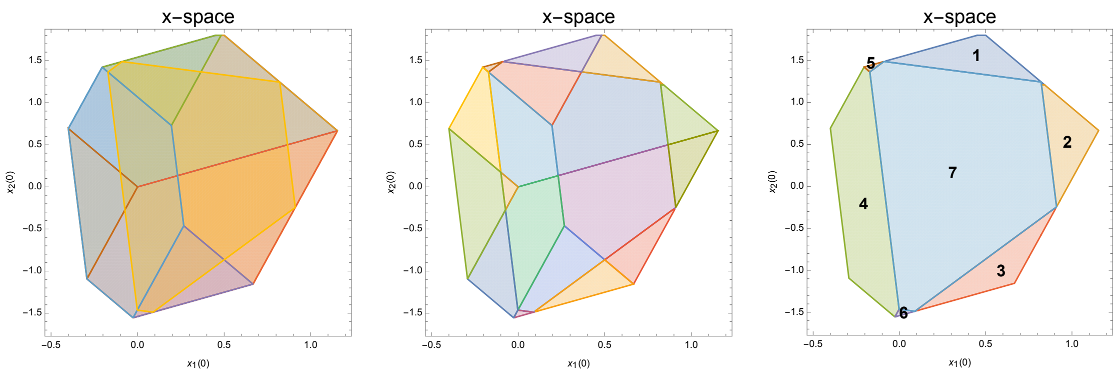
which, again, can be calculated via quantifier elimination using Weispfenning's algorithm. Analogously, let I_1 be the set of states x for which (3.93) will lose feasibility after a single time step:

$$I_1 \equiv \exists u \left(u_0^*(x) \wedge x \in F_0 \wedge Ax + Bu \notin F_0 \right). \tag{3.137}$$

If $I_1 = \varnothing$, the control scheme is said to be recursively feasible (i.e., for all initially feasible states $x(0)$ and for all optimal input sequences, (3.93) remains feasible for all time steps k).

In general, let $F_k, k = 1, 2, \dots$ denote sets of states for which the optimisation problem (3.93) remains feasible for at least $k+1$ time steps, and $I_k, k = 1, 2, \dots$ sets of states for which (3.93) will lose feasibility after k time steps. Then the sequence of sets $\{F_k\}$ can be computed by applying Weispfenning's quantifier elimination algorithm to

$$F_k \equiv \exists u \left(u_0^*(x) \wedge x \in F_{k-1} \wedge Ax + Bu \in F_{k-1} \right), \; k = 1, 2, \dots \tag{3.138}$$



(a) Overlapping regions $O_1(0)$, ..., $O_8(0)$.

(b) Disjoint regions $D_1(0)$, ..., $D_{18}(0)$.

(c) Explicit MPC state-space partition.

Fig. 3.13 Steps in obtaining explicit MPC solution by using the method based on Weispfenning's quantifier elimination algorithm and developed in Section 3.4.3.

while $I_k = F_0 \cap (F_0 \setminus F_k)$.

Now suppose that an additive disturbance $d(k) \in D$ acts on the system (3.91), i.e.:

$$x(k+1) = Ax(k) + Bu(k) + d(k). \tag{3.139}$$

Then the set of states I_1 that will lose feasibility after a single time step can be found by eliminating quantifiers from the formula:

$$\exists u \exists d \left(u_0^*(x) \wedge d \in D \wedge x \in F_0 \wedge Ax + Bu + d \notin F_0\right). \tag{3.140}$$

Several examples of the computation of highly non-convex semialgebraic sets F_k and I_k by using Weispfenning's algorithm (in both nominal and disturbed cases) will be given in Section 3.4.7.

3.4.5 Computational example: explicit MPC solution for a linear system

In this section, we illustrate how the procedure described in Section 3.4.3 can be used to obtain an explicit MPC solution for a linear time invariant system (3.91) with a quadratic objective and polytopic constraints (3.93)

Consider a linear time-invariant discrete-time system of the form (3.91) with

$$A = \begin{bmatrix} \frac{75}{100} & -\frac{1299}{1000} \\ \frac{1299}{1000} & \frac{75}{100} \end{bmatrix}, \; B = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{3.141}$$

and MPC controller designed using $N = 1$, $Q = I$, $R = 1$, together with the constraints $-1 \le x_1 \le 1$, $-1 \le x_2 \le 1$, $-1 \le u_0 \le 1$, $u_o \le \frac{1}{5} + x_1 + x_2$ (taken from Löfberg (2012)).

The procedure developed in Section 3.4.3 for obtaining the algebraic expression of the explicit MPC solution for the system (3.141) goes as follows. Firstly, after performing quantifier elimination using Weispfenning's quantifier elimination algorithm, we obtain eight overlapping regions $O_1(0), \dots, O_8(0)$, as illustrated in Figure 3.13 (a). Then those overlapping regions are split into 18 disjoint regions $D_1(0), \dots, D_{18}(0)$, as shown in Figure 3.13 (b). Finally, after finding the optimal value function $J(x_1(0), x_2(0))$ in each one of those regions, solving for the optimal input u_0^* and merging the regions $D_i(0), i = 1, \dots, 18$ with the same solution, our quantifier elimination based method obtains the explicit MPC control law (see Figure 3.14) whose state-space partition is illustrated in Figure 3.13 (c).

As expected, the expression of the explicit MPC law obtained (see Figure 3.14) by using the method described in Section 3.4.3 is in alignment with the one obtained via the standard approach of using the Multi-Parametric Toolbox (Herceg et al., 2013), the main difference here being that our method produces the answer using exact algebraic numbers.

3.4.6 Computational example: explicit MPC solution for a nonlinear system

In this section, we illustrate how the procedure described in Section 3.4.3 can be used to obtain an explicit MPC solution for a nonlinear MPC system of the form (3.133) with a quadratic cost and linear constraints.

Consider the Duffing oscillator (for more details, see Section 13.5 in Jordan and Smith (2007)), which is a nonlinear oscillator of the second order

$$\ddot{y}(t) + 2\xi\dot{y}(t) + y(t) + y(t)^3 = u(t) \tag{3.142}$$

with the damping coefficient $\xi = \frac{3}{10}$, $y(t) \in \mathbb{R}$ being the continuous state variable and $u(t) \in \mathbb{R}$ being the control input. By using a forward difference approximation with a sampling

$$u_0^*(x) = \begin{cases} -1 & \text{if } \begin{bmatrix} 1299 & 750 \\ -750 & 1299 \\ -183 & -683 \end{bmatrix} x \leq \begin{bmatrix} 2000 \\ 2000 \\ -1000 \end{bmatrix}, \text{(region \#1)} \\ \begin{bmatrix} -\frac{1299}{1000} & -\frac{750}{1000} \end{bmatrix} x + 1 & \text{if } \begin{bmatrix} 1299 & 750 \\ 2049 & -549 \\ -1116 & -67 \end{bmatrix} x \leq \begin{bmatrix} 2000 \\ 2000 \\ -1000 \end{bmatrix}, \text{(region \#2)} \\ \begin{bmatrix} \frac{750}{1000} & -\frac{1299}{1000} \end{bmatrix} x - 1 & \text{if } \begin{bmatrix} -933 & 616 \\ 2049 & -549 \\ 750 & -1299 \end{bmatrix} x \leq \begin{bmatrix} -1000 \\ 2000 \\ 2000 \end{bmatrix}, \text{(region \#3)} \\ \begin{bmatrix} \frac{2049}{1000} & -\frac{549}{1000} \end{bmatrix} x + \frac{1}{5} & \text{if } \begin{bmatrix} 1299 & 750 \\ -1116 & -67 \\ 2049 & -549 \\ -683 & 183 \\ -433 & -250 \\ 1116 & 67 \end{bmatrix} x \leq \begin{bmatrix} 800 \\ 400 \\ 800 \\ 400 \\ 400 \\ -100 \end{bmatrix}, \text{(region \#4)} \\ \begin{bmatrix} -\frac{750}{1000} & -\frac{1299}{1000} \end{bmatrix} x + 1 & \text{if } \begin{bmatrix} -750 & 1299 \\ -1299 & -750 \\ 933 & -616 \end{bmatrix} x \leq \begin{bmatrix} 2000 \\ -800 \\ -1000 \end{bmatrix}, \text{(region \#5)} \\ 1 & \text{if } \begin{bmatrix} -2049 & 549 \\ 750 & -1299 \\ 183 & 683 \end{bmatrix} x \leq \begin{bmatrix} -800 \\ 2000 \\ -1000 \end{bmatrix}, \text{(region \#6)} \\ \begin{bmatrix} -\frac{183}{1000} & -\frac{683}{1000} \end{bmatrix} x & \text{if } \begin{bmatrix} 1116 & 67 \\ -933 & 616 \\ -183 & -683 \\ 183 & 683 \\ -1116 & -67 \\ 933 & -616 \end{bmatrix} x \leq \begin{bmatrix} 1000 \\ 1000 \\ 1000 \\ 1000 \\ 100 \\ 1000 \end{bmatrix}, \text{(region \#7)} \end{cases}$$

Fig. 3.14 Explicit MPC solution obtained using the method based on Weispfenning's quantifier elimination algorithm and developed in Section 3.4.3.

period $h = \frac{1}{20}$, we obtain a discrete time state space model

$$\begin{bmatrix} x_1(i+1) \\ x_2(i+1) \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{20} \\ -\frac{1}{20} & \frac{97}{100} \end{bmatrix} \begin{bmatrix} x_1(i) \\ x_2(i) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{20} \end{bmatrix} u(i) + \begin{bmatrix} 0 \\ -\frac{1}{20}x_1^3(i) \end{bmatrix}, \tag{3.143}$$

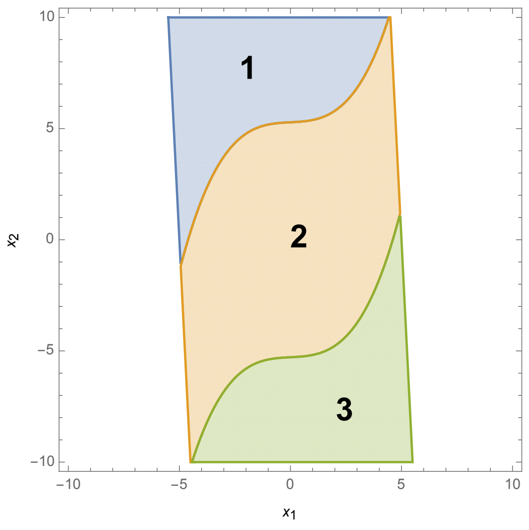
$$y(i) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(i) \\ x_2(i) \end{bmatrix}, \tag{3.144}$$

which is linear in the input $u(i)$ and nonlinear in the state $x_1(i)$. It is important to note that the input-affine system dynamics (3.143) is needed in order for the approach discussed in Section 3.4.3 to be applicable. This restriction on the system dynamics ensures that the MPC problem of interest can be expressed as an equivalent quantifier elimination problem where all quantified variables appear at most quadratically. This in turn allows us to utilise Weispfenning's quantifier elimination algorithm which is the central part of the procedure developed in Section 3.4.3.

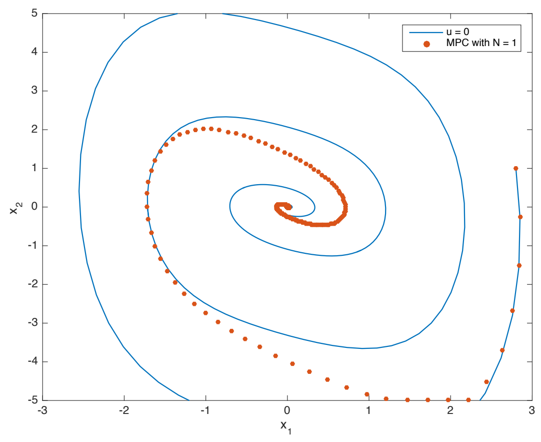
Suppose that the MPC controller is designed with a prediction horizon $N = 1$ and weights $Q = I, R = 1/10$, together with the constraints $|x_1(1)| \le 5, |x_2(1)| \le 5$. Then application of the procedure described in Section 3.4.3 produces the explicit MPC control law $u_0^*(x)$

$$u_0^*(x) = \begin{cases} \frac{1}{5}(500 + 5x_1 + 5x_1^3 - 97x_2) & \text{if } \begin{bmatrix} 20 & 1 & 0 \\ -20 & -1 & 0 \\ 5 & -97 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_1^3 \end{bmatrix} \le \begin{bmatrix} 100 \\ 100 \\ -1025/2 \end{bmatrix}, \\ & \text{(region \#1)} \\ \frac{1}{205}\left(5x_1^3 + 5x_1 - 97x_2\right) & \text{if } \begin{bmatrix} 5 & -97 & 5 \\ -5 & 97 & -5 \\ 20 & 1 & 0 \\ -20 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_1^3 \end{bmatrix} \le \begin{bmatrix} 1025/2 \\ 1025/2 \\ 100 \\ 100 \end{bmatrix}, \\ & \text{(region \#2)} \\ \frac{1}{5}(-500 + 5x_1 + 5x_1^3 - 97x_2) & \text{if } \begin{bmatrix} 20 & 1 & 0 \\ -20 & -1 & 0 \\ -5 & 97 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_1^3 \end{bmatrix} \le \begin{bmatrix} 100 \\ 100 \\ -1025/2 \end{bmatrix} \\ & \text{(region \#3)} \end{cases} \tag{3.145}$$

whose state space partition is given in Figure 3.15 (a). Figure 3.15 (b) illustrates the state space evolution of the system, with and without ($u = 0$) the MPC controller, from $x(0) = \begin{bmatrix} \frac{14}{5} & 1 \end{bmatrix}^T$.



(a) Explicit MPC state-space partition.



(b) State space evolution of the system.

Fig. 3.15 Explicit MPC solution for a nonlinear system.

3.4.7 Computational example: checking recursive feasibility for nominal and disturbed linear systems

In this section, we show how Weispfenning's algorithm (together with the algebraic expression for the explicit MPC solution obtained by the quantifier elimination based method developed in Section 3.4.3) can be used to eliminate quantifiers from various quantified formulas discussed in Section 3.4.4 and, consequently, verify recursive feasibility of the MPC system.

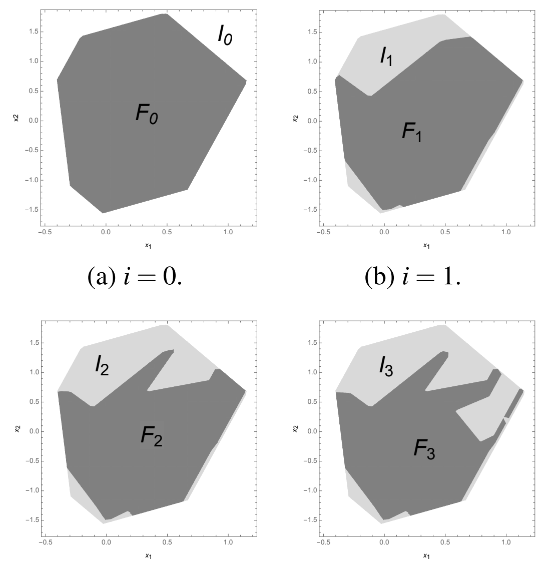
Recursive feasibility — nominal case

Is controller represented in Figure 3.14 recursively feasible? By eliminating quantifiers from the first order formula (3.137), we find that $I_1 \neq \varnothing$, as can be seen in Figure 3.16 (b). Hence, recursive feasibility is violated. Moreover, by repeated application of quantifier elimination, we are able to calculate sets of states I_i which will lose feasibility after i steps (see Figure 3.16).

Recursive feasibility — disturbed case

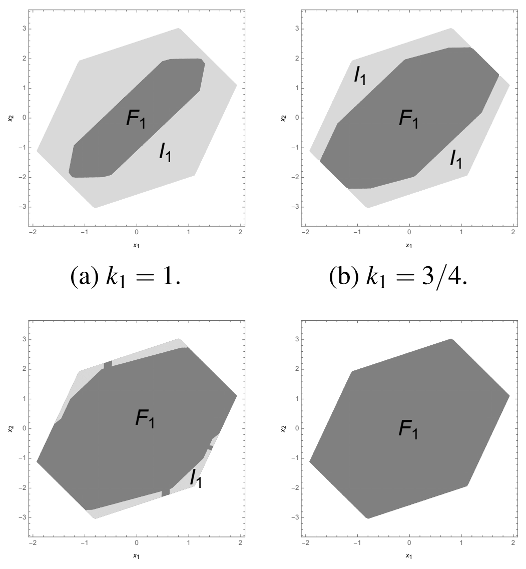
Consider a nominal system of the form $x(i+1) = Ax(i) + Bu(i)$ (3.91) with

$$A = \begin{bmatrix} \frac{45}{100} & -\frac{7794}{10000} \\ \frac{7794}{10000} & \frac{45}{100} \end{bmatrix}, \; B = \begin{bmatrix} -1 \\ 1 \end{bmatrix}. \tag{3.146}$$



(a) $i = 0$. (b) $i = 1$. (c) $i = 2$. (d) $i = 3$.

Fig. 3.16 Sets of states I_i which will lose feasibility after i steps.



(a) $k_1 = 1$. (b) $k_1 = 3/4$. (c) $k_1 = 1/2$. (d) $k_1 = 1/4$.

Fig. 3.17 Sets of states I_1 which will lose feasibility after a single time step.

An MPC controller is designed for this system using prediction horizon $N = 3$, weights $Q = I$, $R = 1$, together with constraints $-1 \leq x_{i+1} \leq 1$, $-1 \leq u_i \leq 1$, $i = 0, 1, 2$ (example taken from Löfberg (2012)). The optimal explicit MPC solution is obtained for this system using the procedure developed in Section 3.4.3. By using quantifier elimination as described in Section 3.4.4, we find that this controller is recursively feasible.

Now suppose that a disturbance $d(i) \in D$ acts on the system:

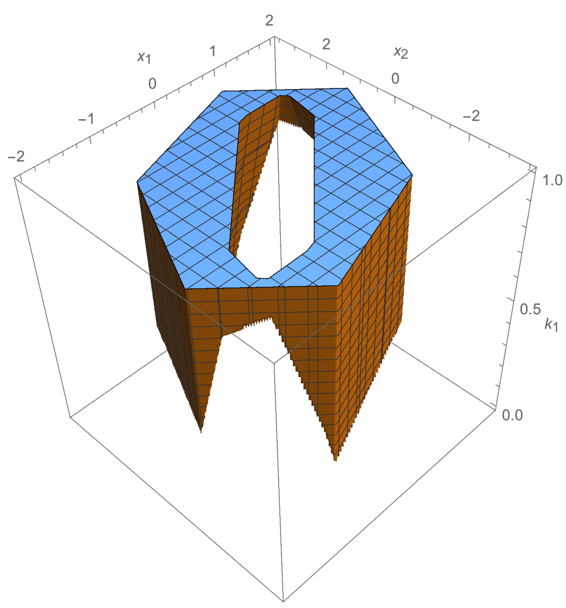
$$x(i+1) = Ax(i) + Bu(i) + d(i). \tag{3.147}$$

Let:

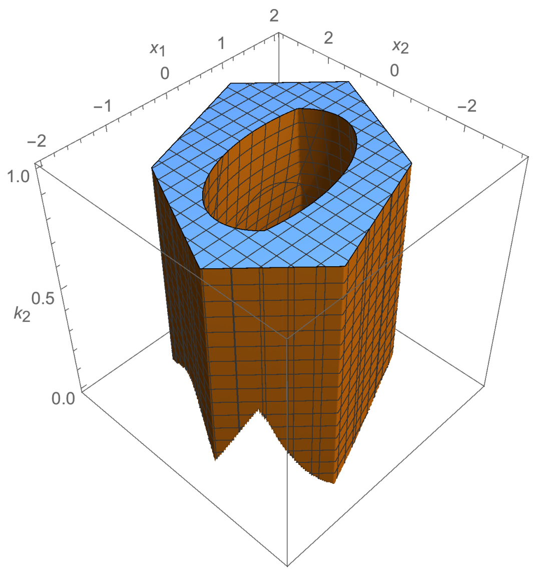
$$D = \left\{ \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \in \mathbb{R}^2 : -k_1 \leq d_1 \leq k_1, -k_1 \leq d_2 \leq k_1, k_1 \geq 0 \right\}. \tag{3.148}$$

For a given value of k_1, which states will lose feasibility after a single time step? By posing this question as a quantifier elimination problem in the form (3.140), and then solving it with an elimination algorithm, we are able to find sets of states I_1 of interest, as depicted in Figure 3.17.

Hence, we see that with $k_1 = 1/4$, $I_1 = \varnothing$, and the system is recursively feasible for all allowable disturbances. Moreover, by keeping k_1 as a variable in (3.140) (rather than assigning a particular numerical value beforehand), and then applying Weispfenning's algorithm,



(a) $0 \leq k_1 \leq 1$, fixed disturbance set.



(b) $0 \leq k_2 \leq 1$, state-dependent disturbance set.

Fig. 3.18 Dependence of the set I_1 on the uncertainty parameter $k_i, i = 1, 2$.

we obtain the dependence of the set I_1 on the magnitude of the disturbance k_1, as depicted in Figure 3.18 (a). As the magnitude of the disturbance k_1 gets larger, more and more states will lose feasibility after a single time step, which is to be expected.

Now consider a state-dependent disturbance:

$$D = \left\{ \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \in \mathbb{R}^2 : d_1^2 + d_2^2 \leq k_2(x_1^2 + x_2^2) \right\}. \tag{3.149}$$

Despite the fact that two quantified variables (d_1 and d_2) appear quadratically in (3.140), Weispfenning's algorithm is capable of eliminating both of them and hence producing the dependence of the set I_1 on the parameter k_2, as illustrated in Figure 3.18 (b).

In conclusion, repeated computation of sets F_k and I_k, in both nominal and disturbed cases, illustrated the capability of Weispfenning's quantifier elimination algorithm to produce analytic expressions of highly non-convex semialgebraic sets. Moreover, all the examples were computed in fractions of a second, whereas attempts to use cylindrical algebraic decomposition (using Mathematica (Wolfram Research Inc., 2016)) were abandoned after about a day, with relentless use of memory.

3.5 Application of formal methods to Linear Temporal Logic specifications

3.5.1 Feasible parameter set calculation: from Linear Temporal Logic specification to quantifier elimination problem

Linear Temporal Logic (LTL) is a logical formalism that extends propositional logic by adding temporal operators that allow a very intuitive way to describe verification requirements based on how the state of the discrete time dynamical system evolves over time. Any LTL formula ψ is built up recursively via the syntax

$$\psi ::= \text{true} \mid \sigma \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid \bigcirc \psi \mid \psi_1 \,\mathscr{U}\, \psi_2, \tag{3.150}$$

where σ are atomic propositions describing the state of the system, $\wedge$ represents logical conjunction, $\neg$ represents logical negation, $\bigcirc$ (pronounced “next”) and $\mathscr{U}$ (pronounced “until”) are temporal operators.

Let $k_c, k_c \in \mathbb{N}_0$ denote the current time step. LTL formula $\bigcirc\psi$ holds at the current time step k_c if ψ holds at the next time step $k_c + 1$. LTL formula $\psi_1 \,\mathscr{U}\, \psi_2$ holds at the current time step k_c, if there is some future time step $k_f, k_f > k_c, k_f \in \mathbb{N}$ at which ψ_2 holds and ψ_1 holds at all time steps until the future time step k_f. For more details on this, see Chapter 5 in (Baier et al., 2008).

In the following, we show how calculation of a feasible parameter set $\Phi(p_1,\dots,p_l)$ for which satisfaction of some particular property of the dynamical system is guaranteed can be expressed as an LTL specification and consequently converted to a quantifier elimination problem of the form discussed in Section 2.3.1. Consider a discrete-time LTI system

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k), \\ y(k) &= C(p_1,\dots,p_l)x(k), \end{aligned} \tag{3.151}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the input, $y \in \mathbb{R}^q$ is the output with an initial state $x(0) \in \mathbb{X}_{ver}$, bounds on the input $u(k) \in \mathbb{U}_{ver}$ $\forall k = 0,1,2,\dots$ and $p_1,\dots,p_l$ denote the parameters parametrising the output.

Suppose that the *k-th* output, $y(k)$, is required to stay in a particular set, $y(k) \in \mathbb{Y}_{ver}$. This requirement can be expressed as an LTL formula:

$$\psi_1 \equiv \bigcirc^k y(0) \in \mathbb{Y}_{ver}. \tag{3.152}$$

Consequently, calculation of a feasible parameter set $\Phi_1(p_1,\ldots,p_l)$ for which (3.152) holds for the system (3.151) can be expressed as a quantifier elimination problem:

$$\Phi_1(p_1,\ldots,p_l) \equiv \forall u(0),u(1),\ldots,u(k-1), \bigwedge_{i=0}^{k-1} u(i) \in \mathbb{U}_{ver} \implies \left(C(p_1,\ldots,p_l)A^k x(0) + \sum_{l=0}^{k-1} CA^{k-1-l}Bu(l)\right) \in \mathbb{Y}_{ver}. \tag{3.153}$$

Similarly, consider the requirement for the output $y(k)$ to stay in a particular set, $y(k) \in \mathbb{Y}_{ver}$, for the next k time steps. An equivalent LTL formula representing this requirement is given by:

$$\psi_2 \equiv \bigwedge_{i=1}^{k} \left(\bigcirc^i y(0) \in \mathbb{Y}_{ver}\right). \tag{3.154}$$

Analogously, calculation of a feasible parameter set $\Phi_2(p_1,\ldots,p_l)$ for which (3.154) holds for the system (3.151) can be expressed as a quantifier elimination problem:

$$\Phi_2(p_1,\ldots,p_l) \equiv \forall u(0),u(1),\ldots,u(k-1), \bigwedge_{i=0}^{k-1} u(i) \in \mathbb{U}_{ver} \implies \bigwedge_{i=1}^{k} \left(C(p_1,\ldots,p_l)A^i x(0) + \sum_{l=0}^{i-1} CA^{i-1-l}Bu(l)\right) \in \mathbb{Y}_{ver}. \tag{3.155}$$

In case $\mathbb{U}_{ver}$ is a polytope, all quantified variables in (3.153) and (3.155) (which are inputs in this case) appear linearly and hence can be eliminated by applying Weispfenning's virtual substitution algorithm (Loos and Weispfenning, 1993; Weispfenning, 1997) discussed in Section 2.3.2.

Feasible parameter sets (3.153) and (3.155) are calculated in Haesaert et al. (2017) via the state of the art method that is essentially equivalent to a reachability algorithm. This, consequently, results in the following restrictions:

- Restricted fragment of LTL that excludes negation. The method used in Haesaert et al. (2017) for calculating $\Phi_i(p_1,\ldots,p_l)$, $i = 1,2$ can't deal with negation because it relies on the assumption of convexity of the corresponding feasible parameter set. Having both negation and conjunction in the logic simultaneously allows disjunction (assuming ψ_1 and ψ_2 are two LTL specifications, $\psi_1 \vee \psi_2 \equiv \neg(\neg\psi_1 \wedge \neg\psi_2)$) which in turn allows non-convex feasible parameter sets to arise.

- Only linearly parametrized dynamical systems are allowed in order to ensure that the resulting feasible parameter set is polyhedral.

Both of those restrictions can be avoided by converting an LTL specification to a quantifier elimination problem and then calculating the feasible parameter set $\Phi_i(p_1,\dots,p_l)$, $i=1,2$ by eliminating quantifiers with one of the quantifier elimination algorithms, consequently allowing us to extend the results obtained in Haesaert et al. (2017). An illustrative example is provided in Section 3.5.2.

3.5.2 Computational examples

Consider the system of the form

$$x(k+1) = \begin{bmatrix} a & 0 \\ 1-a^2 & a \end{bmatrix} x(k) + \begin{bmatrix} \sqrt{1-a^2} \\ -a\sqrt{1-a^2} \end{bmatrix} u(k), \qquad (3.156)$$
$$y(k) = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}^T x(k),$$

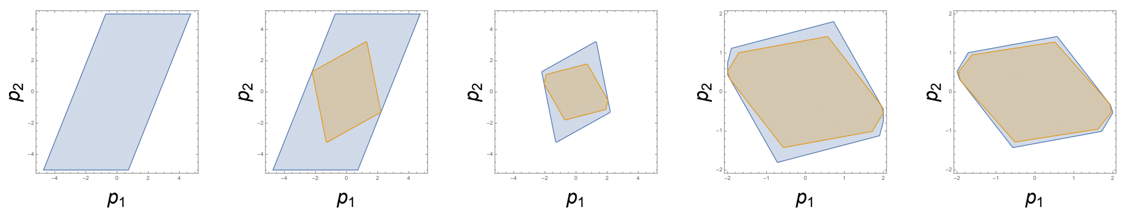
with $|a| \leq 1$ (taken from Haesaert et al. (2017)).

Bounded-time safety verification in restricted fragment of LTL that excludes negation Consider the fragment of LTL (3.150) that excludes negation, i.e. LTL formula ψ is built up recursively via the syntax:

$$\psi ::= \text{true} \mid \sigma \mid \psi_1 \wedge \psi_2 \mid \bigcirc\, \psi \mid \psi_1\, \mathscr{U}\, \psi_2. \qquad (3.157)$$

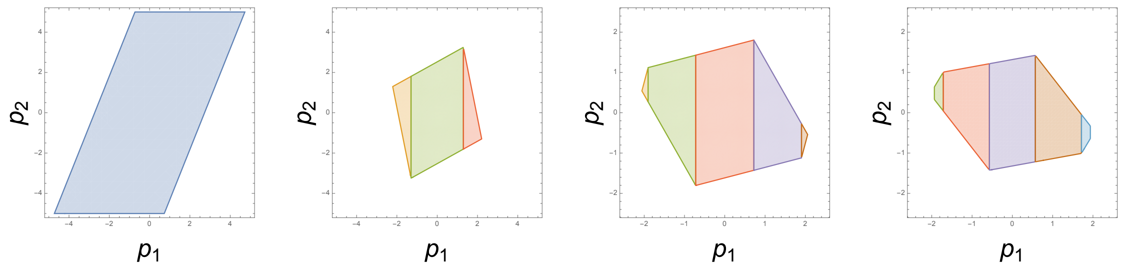
This, in turn, guarantees that feasible parameter sets $\Phi_1(p_1,\dots,p_l)$ and $\Phi_2(p_1,\dots,p_l)$ are convex (see Theorem 2 in Haesaert et al. (2017)). Hence, the method developed in Haesaert et al. (2017) for calculating these sets is still applicable. Therefore, in this section, we show how those feasible parameter sets can be calculated alternatively with quantifier elimination methods.

Let $a=0.4$ in (3.156) with $x(0)\in\{0_2\}=\mathbb{X}_{ver}$, $\mathbb{U}_{ver}=[-0.2,0.2]$ and $\mathbb{Y}_{ver}=[-0.5,0.5]$ and calculate $\Phi_2(p_1,p_2)$ (3.155) for $k=1,2,,\dots,5$. The corresponding feasible sets calculated via Weispfenning's quantifier elimination algorithm are shown in Figure 3.19. The feasible parameter sets $\Phi_2(p_1,p_2)$ obtained by Weispfenning's algorithm are expressed as unions of overlapping polytopic regions — Figure 3.19 represents the unions corresponding to $\Phi_2(p_1,p_2)$. For example, for $k=1,2$, the feasible parameter sets are:



(a) $k = 1$ (blue). (b) $k = 1$ (blue), $k = 2$ (orange). (c) $k = 2$ (blue), $k = 3$ (orange). (d) $k = 3$ (blue), $k = 4$ (orange). (e) $k = 4$ (blue), $k = 5$ (orange).

Fig. 3.19 Feasible sets of parameters $\Phi_2(p_1, p_2)$ for $k = 1, 2, \dots, 5$ calculated by Weispfenning's quantifier elimination algorithm. The set of feasible parameters $\Phi_2(p_1, p_2)$ contracts as k increases.



(a) $k = 1$. (b) $k = 2$. (c) $k = 3$. (d) $k = 4$.

Fig. 3.20 Feasible sets of parameters $\Phi_2(p_1, p_2)$ for $k = 1, 2, 3, 4$ obtained with CAD algorithm.

$$\Phi_2(p_1, p_2),\ k = 1 : 5p_1 - 2p_2 - \frac{25000}{1833} \leq 0 \wedge -5p_1 + 2p_2 - \frac{25000}{1833} \leq 0,$$
$$\Phi_2(p_1, p_2),\ k = 2 : -\frac{125000}{5499} \leq 5p_1 - 9p_2 \leq \frac{125000}{5499} \wedge -\frac{125000}{12831} \leq 5p_1 + p_2 \leq \frac{125000}{12831}.$$

Additionally, we calculate $\Phi_2(p_1, p_2)$ (3.155) for $k = 1, 2, , \dots, 5$ by using CAD. The output of this algorithm gives feasible parameter sets $\Phi_2(p_1, p_2)$ as a union of disjoint polytopes, as depicted in Figure 3.20 (for case $k = 5$, CAD algorithm did not produce an answer in 6 hours). As expected, the feasible parameter sets obtained by using Weispfenning's algorithm (and depicted in Figure 3.19) and by using CAD algorithm (and depicted in Figure 3.20) are the same. Hence, the main advantage of CAD over Weispfenning's algorithm is that it produces an answer which is much simpler (i.e., a union of disjoint polytopic regions rather than a union of overlapping polytopic regions).

Bounded-time safety verification with negation allowed in the logic

Now consider the full LTL (3.150), with the logical negation $\neg\psi$ operation allowed in the logic. Consequently, this allows non-convex feasible parameter sets $\Phi_1(p_1, \dots, p_l)$ and

$\Phi_2(p_1,\ldots,p_l)$ to arise and therefore the method developed in Haesaert et al. (2017) is no longer applicable. Hence, in this section, the advantage of the quantifier elimination based method is illustrated.

Example 1

Let $a = 0.4$ in (3.156) initialised with $\mathbb{X}_{ver} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ and bounds on the input $\mathbb{U}_{ver} = [-u_b, u_b], u_b > 0$. Define atomic propositions σ_1 and σ_2 as $\sigma_1 \leftrightarrow \{y(0) \leq 0\}$, $\sigma_2 \leftrightarrow \{y(0) \leq -1\}$, respectively, and consider the LTL specification:

$$\psi(k) = O^k \neg(\sigma_1 \wedge \neg\sigma_2). \tag{3.158}$$

Calculation of the feasible parameter sets $\Phi_1(p_1,p_2)$ for which (3.158) holds can be expressed as a quantifier elimination problem (3.153) with:

$$\mathbb{Y}_{ver} = (-\infty,-1] \cup (0,+\infty). \tag{3.159}$$

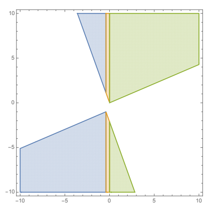
The feasible parameter sets for $k = 1,2,\ldots,5$ and $u_b = 1, 1/2, 1/4, 1/8$, obtained via CAD, are shown in Figure 3.21. Empty figures represent an empty feasible parameter set and different colours in those plots correspond to disjoint regions obtained by the the CAD algorithm. For example, for $u_b = 1/8$ and $k = 3$, the feasible set of parameters is given by:

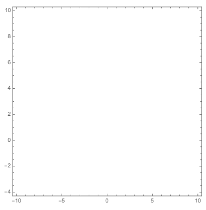
$$\begin{aligned}\Phi_1(p_1,p_2) \equiv & \left(p_1 \leq -\frac{800000}{511163} \wedge \left(p_2 < \frac{229435p_1 - 2000000}{602937} \vee p_2 > -\frac{37465p_1}{66477}\right)\right) \vee \\ & \left(-\frac{800000}{511163} < p_1 \leq 0 \wedge \left(p_2 < \frac{-228815p_1 - 2000000}{419637} \vee p_2 > -\frac{37465p_1}{66477}\right)\right) \vee \\ & \left(0 < p_1 \leq \frac{6800000}{61303} \wedge \left(p_2 < \frac{-228815p_1 - 2000000}{419637} \vee p_2 > \frac{229435p_1}{602937}\right)\right) \vee \\ & \left(p_1 > \frac{6800000}{61303} \wedge \left(p_2 < \frac{-412115p_1 - 2000000}{731247} \vee p_2 > \frac{229435p1}{602937}\right)\right).\end{aligned}$$

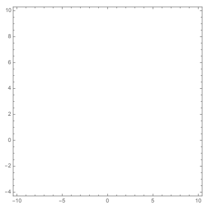
Using Weispfenning's algorithm for quantifier elimination produces an equivalent result and represents it as a union of overlapping polytopes.

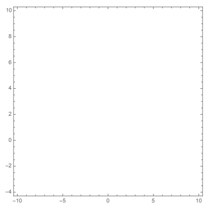
Example 2

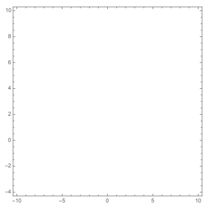
Let $a = 0.4$ in (3.156) initialised with $\mathbb{X}_{ver} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ and bounds on the input $\mathbb{U}_{ver} =$

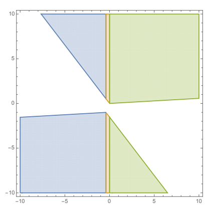

(a) $u_b = 1$ and $k = 1$.

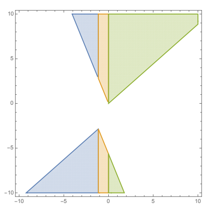

(b) $u_b = 1$ and $k = 2$.

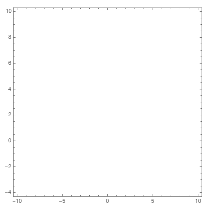

(c) $u_b = 1$ and $k = 3$.

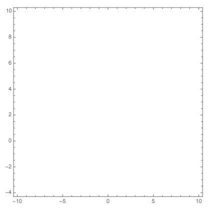

(d) $u_b = 1$ and $k = 4$.

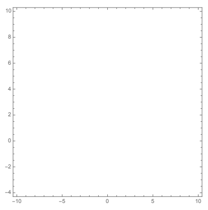

(e) $u_b = 1$ and $k = 5$.

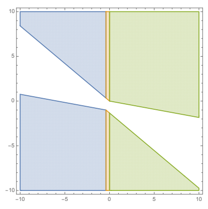

(f) $u_b = 1/2$ and $k = 1$.

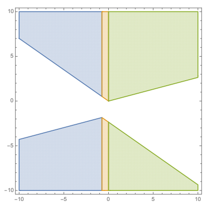

(g) $u_b = 1/2$ and $k = 2$.

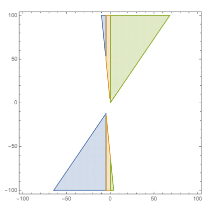

(h) $u_b = 1/2$ and $k = 3$.

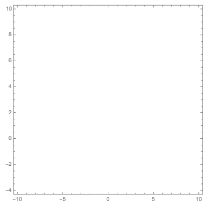

(i) $u_b = 1/2$ and $k = 4$.

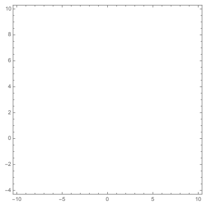

(j) $u_b = 1/2$ and $k = 5$.

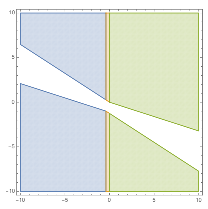

(k) $u_b = 1/4$ and $k = 1$.

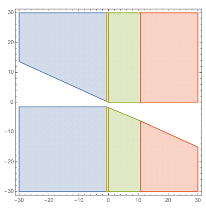

(l) $u_b = 1/4$ and $k = 2$.

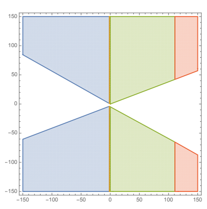

(m) $u_b = 1/4$ and $k = 3$.

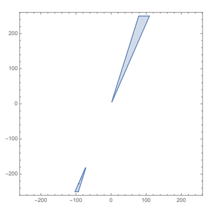

(n) $u_b = 1/4$ and $k = 4$.


(o) $u_b = 1/4$ and $k = 5$.


(p) $u_b = 1/8$ and $k = 1$.


(q) $u_b = 1/8$ and $k = 2$.


(r) $u_b = 1/8$ and $k = 3$.


(s) $u_b = 1/8$ and $k = 4$.

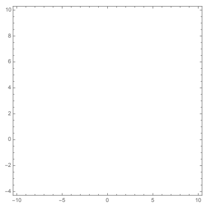

(t) $u_b = 1/8$ and $k = 5$.

Fig. 3.21 Feasible sets of parameters $\Phi_1(p_1, p_2)$ for different values of u_b and k.

$[-u_b, u_b], u_b > 0$. Define a set of atomic propositions

$$\begin{aligned}
&\sigma_1 \leftrightarrow \{y(0) \leq -3\}, \sigma_2 \leftrightarrow \{y(0) \leq -2\},\\
&\sigma_3 \leftrightarrow \{y(0) \leq -1\}, \sigma_4 \leftrightarrow \{y(0) \leq 0\},\\
&\sigma_5 \leftrightarrow \{y(0) \leq 1\}, \sigma_6 \leftrightarrow \{y(0) \leq 2\},
\end{aligned}$$

and consider the LTL specification:

$$\psi(k) = O^k(\neg\sigma_1 \wedge \sigma_2) \vee (\neg\sigma_3 \wedge \sigma_4) \vee (\neg\sigma_5 \wedge \sigma_6). \tag{3.160}$$

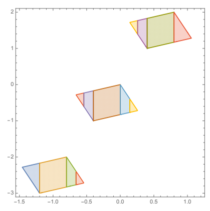
Calculation of the feasible parameter sets $\Phi_1(p_1, p_2)$ for which (3.160) holds can be expressed as a quantifier elimination problem (3.153) with:

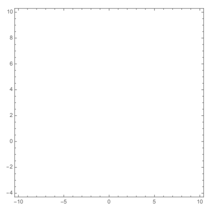
$$\mathbb{Y}_{ver} = (-3,-2] \cup (-1,0] \cup (1,2]. \tag{3.161}$$

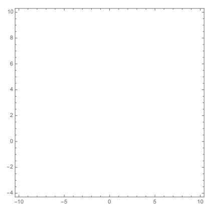
The feasible parameter sets for $k = 1, 2, \ldots, 5$ and $u_b = 1, 1/2, 1/4, 1/8$, obtained via CAD, are shown in Figure 3.22. For example, for $u_b = 1$ and $k = 1$, the feasible set of parameters is given by:

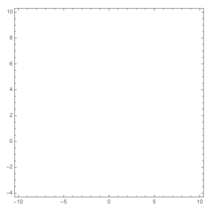
$$\begin{aligned}
&\Phi_1(p_1, p_2) \equiv\\
&\left(-\frac{891}{611} < p_1 \leq -\frac{6}{5} \wedge \frac{-13165p_1 - 30000}{4734} < p_2 < \frac{5165p_1 - 20000}{12066}\right) \vee\\
&\left(-\frac{6}{5} < p_1 \leq -\frac{4}{5} \wedge \frac{5165p_1 - 30000}{12066} < p_2 < \frac{5165p_1 - 20000}{12066}\right) \vee\\
&\left(-\frac{4}{5} < p_1 \leq -\frac{2011}{3055} \wedge \frac{5165p_1 - 30000}{12066} < p_2 < \frac{-13165p_1 - 20000}{4734}\right) \vee\\
&\left(-\frac{2011}{3055} < p_1 \leq -\frac{331}{611} \wedge \left(\frac{5165p_1 - 30000}{12066} < p_2 < \frac{-13165p_1 - 20000}{4734} \vee \frac{-13165p_1 - 10000}{4734} < p_2 < \frac{5165p_1}{12066}\right)\right) \vee\\
&\left(-\frac{331}{611} < p_1 \leq -\frac{2}{5} \wedge \frac{-13165p_1 - 10000}{4734} < p_2 < \frac{5165p_1}{12066}\right) \vee\\
&\left(-\frac{2}{5} < p_1 \leq 0 \wedge \frac{5165p_1 - 10000}{12066} < p_2 < \frac{5165p_1}{12066}\right) \vee\\
&\left(0 < p_1 \leq \frac{433}{3055} \wedge \frac{5165p_1 - 10000}{12066} < p_2 < -\frac{13165p_1}{4734}\right) \vee\\
&\left(\frac{433}{3055} < p_1 \leq \frac{789}{3055} \wedge \left(\frac{5165p_1 - 10000}{12066} < p_2 < -\frac{13165p_1}{4734} \vee \frac{10000 - 13165p_1}{4734} < p_2 < \frac{5165p_1 + 20000}{12066}\right)\right) \vee\\
&\left(\frac{789}{3055} < p_1 \leq \frac{2}{5} \wedge \frac{10000 - 13165p_1}{4734} < p_2 < \frac{5165p_1 + 20000}{12066}\right) \vee\\
&\left(\frac{2}{5} < p_1 \leq \frac{4}{5} \wedge \frac{5165p_1 + 10000}{12066} < p_2 < \frac{5165p_1 + 20000}{12066}\right) \vee\\
&\left(\frac{4}{5} < p_1 < \frac{3233}{3055} \wedge \frac{5165p_1 + 10000}{12066} < p_2 < \frac{20000 - 13165p_1}{4734}\right).
\end{aligned}$$

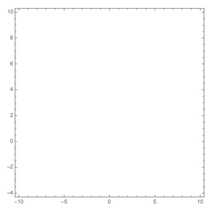
These two examples illustrate the kind of problems that quantifier elimination based method is capable of dealing with while the method presented in Haesaert et al. (2017) is not.

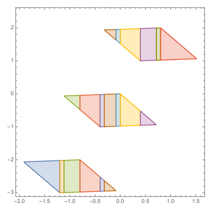

(a) $u_b = 1$ and $k = 1$.

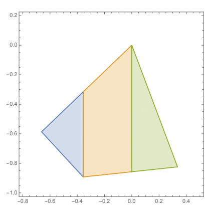

(b) $u_b = 1$ and $k = 2$.

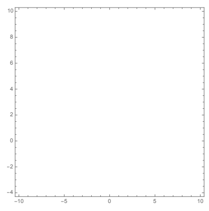

(c) $u_b = 1$ and $k = 3$.

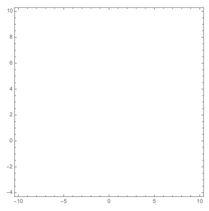

(d) $u_b = 1$ and $k = 4$.

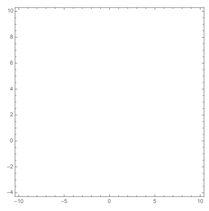

(e) $u_b = 1$ and $k = 5$.

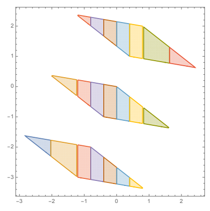

(f) $u_b = 1/2$ and $k = 1$.

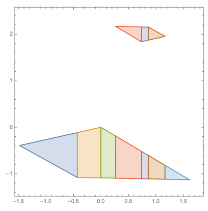

(g) $u_b = 1/2$ and $k = 2$.

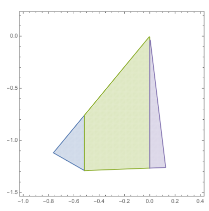

(h) $u_b = 1/2$ and $k = 3$.

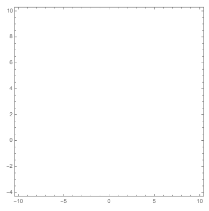

(i) $u_b = 1/2$ and $k = 4$.

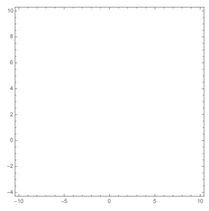

(j) $u_b = 1/2$ and $k = 5$.

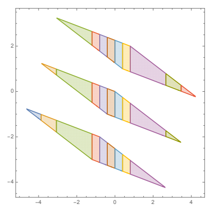

(k) $u_b = 1/4$ and $k = 1$.

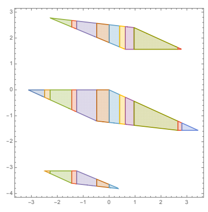

(l) $u_b = 1/4$ and $k = 2$.

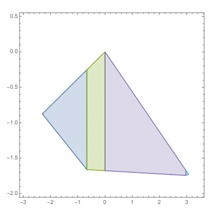

(m) $u_b = 1/4$ and $k = 3$.

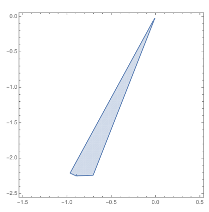

(n) $u_b = 1/4$ and $k = 4$.


(o) $u_b = 1/4$ and $k = 5$.


(p) $u_b = 1/8$ and $k = 1$.


(q) $u_b = 1/8$ and $k = 2$.


(r) $u_b = 1/8$ and $k = 3$.


(s) $u_b = 1/8$ and $k = 4$.

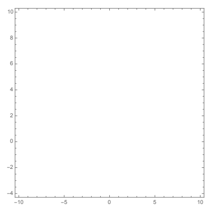

(t) $u_b = 1/8$ and $k = 5$.

Fig. 3.22 Feasible sets of parameters $\Phi_1(p_1, p_2)$ for different values of u_b and k.

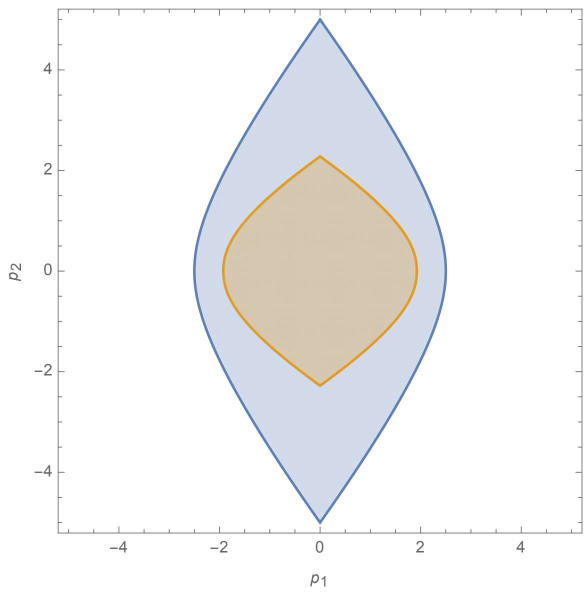


Fig. 3.23 Feasible sets of parameters $\Phi_a(p_1, p_2)$ for $k = 1$ (blue region) and $k = 2$ (orange region).

Nonlinearly parametrised model set

In this section, we show how it is possible to deal with the calculation of the sets of feasible parameters for a nonlinearly parametrised model set which is beyond the applicability of the method developed in Haesaert et al. (2017).

Consider $|a| \leq 1$ as a varying parameter in (3.156) and then let $b = \sqrt{1-a^2}$. Suppose that $x(0) \in \{0_2\} = \mathbb{X}_{ver}$, $\mathbb{U}_{ver} = [-0.2, 0.2]$ and $\mathbb{Y}_{ver} = [-0.5, 0.5]$. Hence, the calculation of the set of feasible parameters $\Phi_a(p_1, p_2)$ for which the output stays in $\mathbb{Y}_{ver}$ within the next k time steps for all systems in the model set parametrised by a can be expressed as a quantifier elimination problem of the form:

$$\begin{aligned}\Phi_a(p_1,p_2) =& \forall a, b, u(0), u(1), \ldots, u(k-1),\\ & \bigwedge_{i=0}^{k-1} (-0.2 \leq u(k) \leq 0.2) \bigwedge (-1 \leq a \leq 1) \bigwedge (a^2 + b^2 = 1) \implies \\ & \bigwedge_{i=1}^{k} \left(-\frac{1}{2} \leq \sum_{l=0}^{i-1} CA^{i-1-l} Bu(l) \leq \frac{1}{2} \right).\end{aligned}$$

Feasible sets of parameters $\Phi_a(p_1, p_2)$ for $k = 1, 2$ are depicted in Figure 3.23 (computation for cases $k \geq 3$ did not finish in 6 hours). In order to eliminate quantifiers, both Weispfenning's virtual substitution and CAD algorithms were used. Weispfenning's algorithm initially removes all quantified inputs and parameter a which results in a first order formula with quantified variable b. Consequently, since parameter b appears with higher than quadratic order (4-th order for $k = 1$ and 6-th order for $k = 2$), CAD has to be used to eliminate it.

For illustration purposes, we provide semialgebraic set $\Phi_a(p_1, p_2)$ for $k = 1$

$$\begin{aligned}
&\left(p_1 = -\frac{5}{2} \wedge p_2 = 0\right) \vee \left(p_1 = \frac{5}{2} \wedge p_2 = 0\right) \vee \\
&-\frac{5}{2} < p_1 < \frac{5}{2} \wedge -\sqrt{\text{Root}\left[4x^3 + x^2\left(-12p_1^2 - 200\right) + x\left(12p_1^4 - 500p_1^2 + 2500\right) - 4p_1^6 + 25p_1^4, 2\right]} \le p_2 \\
&\le \sqrt{\text{Root}\left[4x^3 + x^2\left(-12p_1^2 - 200\right) + x\left(12p_1^4 - 500p_1^2 + 2500\right) - 4p_1^6 + 25p_1^4, 2\right]},
\end{aligned}$$

where $\text{Root}[f(x), k]$ represents the exact k-th root of the polynomial equation $f(x) = 0$ (all real roots come before the complex ones, are ordered in an increasing order, complex conjugate pairs of roots are considered to be adjacent with the root with the negative imaginary part coming first).

3.6 Industrial example: verification of a longitudinal backup flight control law to accommodate sensor loss in the RECONFIGURE benchmark

In this section, we finally use the formal methods based verification framework to verify several properties of a control law developed for a real world industrial system from the aerospace field. Section 3.6.1 gives a short summary of the control law in question. Section 3.6.2 discusses how verification criteria of interest can be cleared at particular flight points, while Section 3.6.3 shows how performance and robustness conditions of interest can be checked throughout the whole flight envelope. In conclusion, this application illustrates that verification frameworks based on formal methods can be used to verify properties of real-world industrial systems.

3.6.1 Description of the control law

In this section, we give a short summary of the backup flight control law developed in Maciejowski et al. (2016) for the RECONFIGURE project (Goupil et al., 2014) that will be used as an example of controller clearance via formal methods. It is a robust vertical load factor (n_z) tracking control law which is based on application of theory from robust MPC and $\mathscr{H}_2$ optimal control. This control law is supposed to be used as a backup to the baseline

Airbus control law in the case when calibrated airspeed (*VCAS*) measurement is lost due to multiple simultaneous sensor failures.

For the purpose of control design, Airbus has provided RECONFIGURE project members with linearisations $A_{sp}(\theta)$, $B_{sp}(\theta)$, $C_{sp}(\theta)$, $D_{sp}(\theta)$ of the longitudinal short period dynamics of an Airbus A380 aircraft at 234 different flight points θ_i, $i \in \{1,\ldots,234\}$ covering a flight envelope of airspeed, altitude, mass, centre of gravity, landing gear and slat/flap positions. At those flight points, short period dynamics are

$$x(k+1) = \begin{bmatrix} q(k+1) \\ \alpha(k+1) \end{bmatrix} = A_{sp}(\theta) \begin{bmatrix} q(k) \\ \alpha(k) \end{bmatrix} + B_{sp}(\theta)u(k), \tag{3.162}$$

$$y(k) = \begin{bmatrix} q(k) \\ n_z(k) \end{bmatrix} = C_{sp}(\theta) \begin{bmatrix} q(k) \\ \alpha(k) \end{bmatrix} + D_{sp}(\theta)u(k), \tag{3.163}$$

where q is pitch rate, α is angle of attack, n_z is vertical load factor (i.e., the acceleration normal to the aircraft body divided by acceleration due to gravity), and u represents elevator deflection.

For controller design purposes, the short period model (3.162) at each flight point is augmented with simplified sensor, filter and actuator models. To avoid the need for estimating unmeasured states using an observer, this augmented model is transformed into a non-minimal realisation whose state vector is comprised of a finite time history of past inputs and outputs. Finally, for tracking, the model of the plant is augmented with an integrator of the error between the load factor to be tracked and the reference. This results in the following augmented system dynamics

$$\bar{x}(k+1) = \bar{A}(\theta)\bar{x}(k) + \bar{B}(\theta)u(k) + B_r r(k), \tag{3.164}$$

where $r(k)$ is the vertical load factor (n_z) reference and $\bar{x}(k) \in \mathbb{R}^{14}$. The state space representation (3.164) is used for control gain synthesis.

Additionally, the 234 flight points θ_i, $i \in \{1,\ldots,234\}$ are split into 55 flight groups J_j, $j \in \{1,\ldots,55\}$. All flight points in a particular flight group have the same mass, altitude, centre of gravity, slat/flap and landing gear positions, but different calibrated airspeed (*VCAS*). The control design objective is to synthesize 55 state-feedback control gains K_j (one for each flight group J_j) that stabilise the augmented model (3.164) at each flight point in the flight group J_j. This is achieved by solving the following semidefinite programming problem formulated for this particular application in Maciejowski et al. (2016) which applies mathematical techniques of Cuzzola et al. (2002), de Oliveira et al. (1999) and de Oliveira et al. (2002).

Proposition 4 *In the following, θ_{ji} denotes the i-th flight point in the j-th flight group J_j. There exists a stabilising state-feedback control law $u(k) = K_j\bar{x}(k) = Y_jG_j{}^{-1}\bar{x}(k)$, meeting the performance requirements:*

- *requirement #1: upper bound on the cost function*

$$\sum_{k=0}^{+\infty} \bar{x}(k)^T \left(Q_j + K_j{}^T R_j K_j + S_j K_j + K_j{}^T S_j^T\right) \bar{x}(k) \leq \gamma_j, \tag{3.165}$$

  where Q_j,R_j and S_j are weighting matrices for the flight group J_j.

- *requirement #2: decrease in the Lyapunov function*

$$V(\bar{x}(k)) = \bar{x}(k)^T \left(\gamma_j X_{ji}^{-1}\right) \bar{x}(k) \tag{3.166}$$

  is at least as big as a stage cost.

These requirements are satisfied if $\forall \theta_{ji} \in J_j$ (i.e. for each flight point in this particular flight group), there exist symmetric matrices X_{ij}, and for each flight group J_j, there exists a pair of matrices $\{Y_j,\ G_j\}$ satisfying the LMIs

$$\begin{bmatrix} G_j + G_j^T - X_{ji} & * & * & * \\ \overline{A}(\theta_{ji})G_j + \overline{B}(\theta_{ji})Y_j & X_{ji} & 0 & 0 \\ (Q_j - S_jR_j{}^{-1}S_j{}^T)^{1/2}G_j & 0 & \gamma_j I & 0 \\ R_j{}^{1/2}(Y_j + R_j{}^{-1}S_j{}^T G_j) & 0 & 0 & \gamma_j I \end{bmatrix} \geq 0, \textit{(requirement \#2)}, \tag{3.167}$$

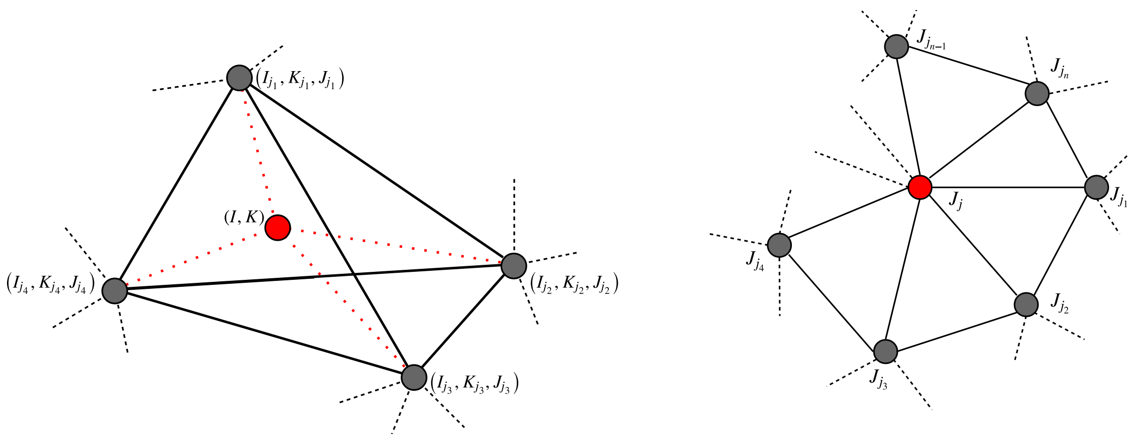
$$\begin{bmatrix} 1 & \bar{x}(k)^T \\ \bar{x}(k) & X_{ji} \end{bmatrix} \geq 0, \textit{(requirement \#1)}, \tag{3.168}$$

where $*$ is used to induce a symmetric structure in the matrix. ■

Hence, the upper bound on the cost function γ_j for the flight group J_j is minimised by solving the following LMI optimisation problem:

$$\min_{\gamma_j,\, G_j,\, X_{ji},\, Y_j} \gamma_j \text{ subject to (3.167) and (3.168).} \tag{3.169}$$

Consequently, a controller designed using Proposition 4 for a given flight group J_j will stabilise any realisation of the system (3.164) in that flight group. However, the flight groups


$(I_{j_1}, K_{j_1}, J_{j_1})$
(I, K)
$(I_{j_4}, K_{j_4}, J_{j_4})$
$(I_{j_2}, K_{j_2}, J_{j_2})$
$(I_{j_3}, K_{j_3}, J_{j_3})$
$J_{j_{n-1}}$
J_{j_n}
J_j
J_{j_1}
J_{j_4}
J_{j_2}
J_{j_3}

(a) Delaunay Triangulation and linear interpolation.

(b) Flight group J_j with its neighbouring flight groups $N_j = \{J_{j_1}, \ldots, J_{j_n}\}$ in Delaunay Triangulation.

Fig. 3.24 Computation of the control law $u(k) = K\bar{x}(k)$.

only correspond to samples of the flight envelope, corresponding to discrete values of mass, altitude, and centre of gravity. In reality, these parameters can also take on values between the samples. Linear interpolation is an approach that can be used to accommodate this practical issue.

The computation of the control gain K in the control law $u(k) = K\bar{x}(k)$ for a general point in the flight envelope is illustrated in Figure 3.24 (a). Consider the case when all flight points in the flight group J_j, $j = 0, \ldots, j_{\max}$ have the same mass M_j, altitude z_j and centre of gravity x_j, but differing calibrated airspeed ($VCAS$). Firstly, Delaunay Triangulation is performed over the set of points $I_j = (M_j, z_j, x_j)$, returning a set of vertices $T_j = \{I_{j_1}, I_{j2}, I_{j3}, I_{j4}\}$, each of which defines a tetrahedral simplex in the flight envelope. Then, given a particular point in the flight envelope $I = (M, z, x)$, the simplex T_j it belongs to is found, and the control gain K for that point is computed via linear interpolation of the gains $K_{j_1}, K_{j_2}, K_{j_3}, K_{j_4}$ synthesised for flight groups $J_{j1}, J_{j2}, J_{j3}, J_{j4}$.

Strictly, this interpolated control law does not guarantee stability, since the Lyapunov function for the flight group J_{j_1}, obtained implicitly while synthesizing control gain K_{j_1}, is not necessarily a Lyapunov function for flight groups J_{j2}, J_{j3}, J_{j4}, and vice versa. The solution to this is to add additional LMIs to the optimisation problem in Proposition 4, guaranteeing that a Lyapunov function for the flight group J_{j_1} is also a Lyapunov function for all "neighbouring" flight groups $N_{j_1} = \{J_{j_{1_1}}, \ldots, J_{j_{1_n}}\}$ that are connected by an edge with J_{j_1} in Delaunay Triangulation. This is formalised in the following proposition from Maciejowski et al. (2016) which relies on mathematical techniques of Cuzzola et al. (2002).

Proposition 5 *Let J_j, $j = 0, \ldots, j_{\max}$ be one of the flight groups with a set of neighbouring flight groups $N_j = \{J_{j_1}, \ldots, J_{j_n}\}$, as shown in Figure 3.24 (b). Also, let the index $j_m k$ denote the k-th flight point in the j_m-th neighbouring flight group ($j_m \in \{j_1, \ldots, j_n\}$). Then, if the optimisation problem*

$$\begin{aligned}
&\min_{\gamma_j, G_j, X_{ji}, Y_j, Q_{j_m k}} \gamma_j \\
&\textit{subject to } (3.167),(3.168)\textit{, and} \\
&\begin{bmatrix} G_j + G_j^T - Q_{j_m k} & * \\ \overline{A}(\theta_{j_m k})G_j + \overline{B}(\theta_{j_m k})Y_j & Q_{j_m k} \end{bmatrix} \geq 0
\end{aligned} \tag{3.170}$$

is feasible, the control law $u(k) = K\overline{x}(k)$ (with $K = Y_j {G_j}^{-1}$) is stabilising for any point in the designed-for flight envelope. ■

It is easy to see that the added LMI condition (3.170) is equivalent to the *requirement #2* in Proposition 4 with Q_j, R_j and S_j set to zero which is equivalent to the requirement that the Lyapunov function must be decreasing at each time step — for more details, see Cuzzola et al. (2002). We will refer to controllers obtained by using Proposition 4 as *Type 1 (T1)* and the ones synthesised by Proposition 5 as *Type 2 (T2)*.

Additionally, the control law discussed in this section is implemented using a limited library (named “SAO” and provided by Airbus) of *Simulink* based blocks. For a detailed account of this implementation, see Appendix B.2 on page 155. This is needed in order to reflect the limitations of the Flight Control Computer (FCC) software coding practices which are often quite restrictive in order to facilitate verification of stringent data integrity and real-time requirements, and to fit with current certification processes. The library used only allows scalar signals and operations and forbids the use of operations such as loops, online optimisation and iterative code. Moreover, the designed controller is a backup control law to be used in the event of multiple simultaneous sensor failures. Therefore, a switching strategy from the nominal fully scheduled control law to the robust backup control law is required. This switching strategy achieves continuity of the control input u at the moment of switching (“bumpless transfer”) by appropriately initialising the state $\overline{x}(k)$ (3.164) based on the past values of the nominal control law. Consequently, this leads to reduced transients at the instant of switching.

Finally, in the following two sections, we will attempt to verify that the backup control law synthesised in this section meets the following performance and robustness requirements:

- Avoidance of the particular M-circle exclusion region by the open loop transfer function, as discussed in Section 3.2.1.
- Simultaneous gain and phase margin of at least 6 dB and 45° is obtained, as discussed in Section 3.2.2.
- Satisfaction of $\mathscr{H}_\infty$-norm bounds on transfer functions from the demanded load factor to vertical load factor and pitch rate.

In Section 3.6.2, we will show how the first two criteria can be cleared at the provided 234 flight points θ_i, $i \in \{1,\dots,234\}$. In Section 3.6.3, we will show how the last two criteria can be cleared throughout the whole flight envelope.

3.6.2 Clearance of verification criteria at the given flight points

In this section, we provide preliminary results of clearance of the backup flight control law (defined in Section 3.6.1) at a set of flight points. The necessary symbolic manipulations required to obtain mathematical expressions of verification criteria are performed using *MATLAB* Symbolic Math Toolbox (The MathWorks Inc., 2012c). The obtained verification requirements are then discharged by using a quantifier elimination algorithm based on cylindrical algebraic decomposition implemented in *Mathematica* (Wolfram Research Inc., 2016).

M-circle exclusion region

In this section, we discuss an application of M-circle exclusion region (see Section 3.2.1) to the backup flight control law defined in Section 3.6.1. We choose an M value of $M = 10^{0.05} \approx 1.122$ because we want the magnitude of the complementary sensitivity function to stay below 1 dB. This results in a circular exclusion region in the Nyquist plane with center $x_c = -4.86$, $y_c = 0$ and radius $r = 4.33$. Hence, if the open loop response $L(j\omega)$ does not enter this M-circle exclusion region, the following holds:

- The magnitude of the complementary sensitivity transfer function stays below $20\log_{10} M = 1$ dB.
- The multiplicatively perturbed system $\hat{L}(s) = (1+\Delta(s))L(s)$ is internally stable for all perturbations $\Delta(s) \in \mathscr{H}_\infty$ such that $||\Delta(s)||_\infty < 10^{-0.05}$.

The results of verification are summarised in Table 3.4 (flight points not mentioned in this table passed the verification criterion).

Flight group	Flight point	Type of controller
7	34	*T1 / T2*
8	39	*T1 / T2*
9	44	*T1 / T2*
13	64	*T1 / T2*
14	69	*T2*
16	78	*T1 / T2*
17	83	*T1*
25	121	*T1 / T2*
30	145	*T2*
31	150	*T1 / T2*
32	155	*T2*
34	164	*T1 / T2*
35	169	*T1*
45	212	*T1 / T2*
46	225	*T1 / T2*

Table 3.4 Complete list of flight points that did not pass $|T(j\omega)| < 1\text{dB}$ verification criterion. In the last column, *T1*, *T2* and *T1 / T2* indicate that the system with the *Type 1*, *Type 2* and both *Type 1* and *Type 2* controllers did not pass the test, respectively. In all cases, these flight points are the ones with the largest calibrated airspeed (*VCAS*) in their respective flight groups.

For illustration purposes, consider flight group 17, and, in particular, flight point 83, with an open loop transfer function with a *Type 1* controller

$$L_{T1}(z) = \frac{0.1901z^{13} - 0.8743z^{12} + 1.69z^{11} - 1.797z^{10} + 1.219z^9 - 0.696z^8 + 0.4327z^7 - 0.1853z^6 - 0.03255z^5 + 0.08937z^4 - 0.04565z^3 + 0.01005z^2 - 0.0007975z - 7.07 \cdot 10^{-06}}{z^{14} - 5.546z^{13} + 13.16z^{12} - 17.37z^{11} + 13.91z^{10} - 6.881z^9 + 2.039z^8 - 0.3287z^7 + 0.02198z^6 - 2.128 \cdot 10^{-15}z^5},$$

and an open loop transfer function with a *Type 2* controller:

$$L_{T2}(z) = \frac{0.2029z^{13} - 0.9269z^{12} + 1.786z^{11} - 1.867z^{10} + 1.15z^9 - 0.5269z^8 + 0.3897z^7 - 0.3234z^6 + 0.08062z^5 + 0.1091z^4 - 0.1096z^3 + 0.04303z^2 - 0.007979z + 0.0005741}{z^{14} - 5.546z^{13} + 13.16z^{12} - 17.37z^{11} + 13.91z^{10} - 6.881z^9 + 2.039z^8 - 0.3287z^7 + 0.02198z^6 - 2.128 \cdot 10^{-15}z^5}.$$

For the first order formula that the quantifier elimination algorithm is required to discharge for the *Type 1* controller, see Appendix B.1 on page 151. From Table 3.4, we see that the quantifier elimination algorithm claims that this flight point does not pass the M-circle criterion with a *T1* controller, but does so with a *T2* one. Indeed, if we look at Nyquist plots for systems in the flight group 17 with *T1* (see Figure 3.25 (a)) and *T2* (see Figure 3.25 (b)) controllers, we see that flight point 83 enters the M-circle exclusion region with a *T1* controller and avoids it with a *T2* one. In Nichols plots (Figure 3.25 (c) and Figure 3.25 (d)), M-circle exclusion regions get converted to complicated shapes which are represented by dotted lines for different values of M. Again, 1 dB exclusion region is entered with a *T1* controller and barely avoided with a *T2* one. Finally, we can look directly at the Bode magnitude plots of the complementary sensitivity function $|T(j\omega)|$ in Figure 3.25 (e) and Figure 3.25 (f) and arrive at the same conclusion.

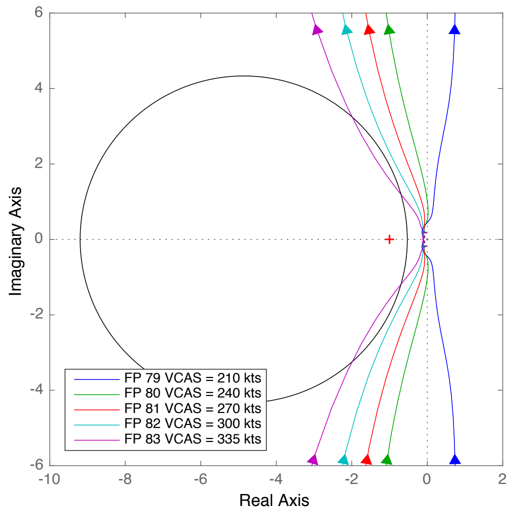
Simultaneous gain and phase margin robustness specification

In this section, we discuss an application of the elliptical Nichols exclusion region as described in Section 3.2.2 to the backup flight control law defined in Section 3.6.1. We choose simultaneous gain and phase margin requirement of $G_m = 6$ dB and $P_m = 45^\circ$, which results in an overbounding circle in the Nyquist plane with centre $(-a,\ 0) = (-1.16,\ 0)$ and radius $r = 0.84$. All 234 flight points passed this criterion with both *T1* and *T2* controllers. In addition to clearing these gain and phase margin criteria, this also implies that, as discussed in Section 3.2.2, for all open loop frequency responses $L_i(j\omega)$, $i \in \{1, \ldots 234\}$ at the provided flight points with both *T1* and *T2* controllers, the following holds:

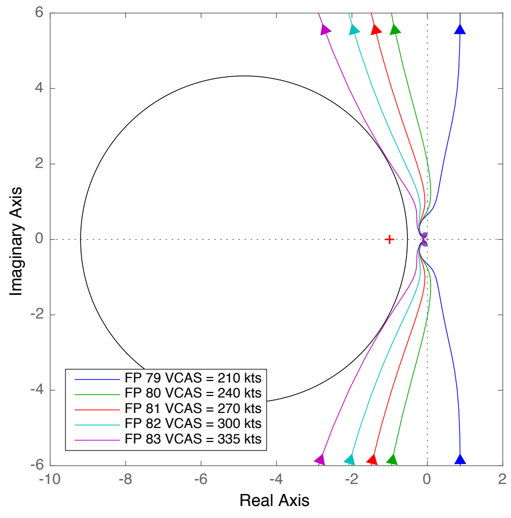
- All multiplicatively perturbed plants with an open-loop transfer function $\hat{L}_i(s) = 1.16 \cdot L_i(s)\left(1 + \frac{21}{29} \cdot \Delta(s)\right)$ with $||\Delta(s)||_\infty < 1$, $\Delta(s) \in \mathscr{H}_\infty$ are closed-loop stable.
- $\left|\left|\frac{1.8125 \cdot L_i(j\omega)}{1+1.8125 \cdot L_i(j\omega)}\right|\right|_\infty < \frac{29}{21}$, where $A = \frac{a}{a^2-r^2} = \frac{1.16}{1.16^2-0.84^2} = 1.8125$ (see (3.24) on page 61).

On the other hand, through repeated visual inspection, it can be found that the control design actually has simultaneous gain and phase margin of $G_m = 6$ dB and $P_m = 60^\circ$ at all flight points with both *T1* and *T2* controllers (i.e., none of the open-loop frequency responses $L(j\omega)$ enter the red elliptical exclusion region shown in Figure 3.26 (a) and Figure 3.26 (c)). According to the results in Section 3.2.2, this elliptical region can be overbounded by a circular exclusion region in the Nyquist plane with centre $(-1,0)$ and radius 1. From Figure 3.26 (b) and Figure 3.26 (d), it is obvious that this overbounding is too conservative, and therefore proving avoidance of exclusion regions directly in the Nichols plane would be desirable. Unfortunately, trying to do so by using the *MetiTarski* theorem prover for real-valued functions for this particular example results in a computationally intractable problem. This additional computational complexity comes from having to repeatedly approximate real-valued functions with appropriate polynomial bounds.

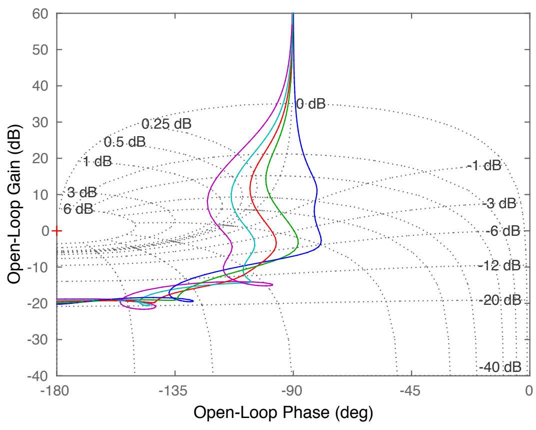
In conclusion, results that were obtained in this section via quantifier elimination algorithm are consistent with the graphical approach. The key difference between the conclusion drawn through applying the proposed formal method and the conclusion drawn by inspecting the frequency domain plots is that the proposed formal approach guarantees the required



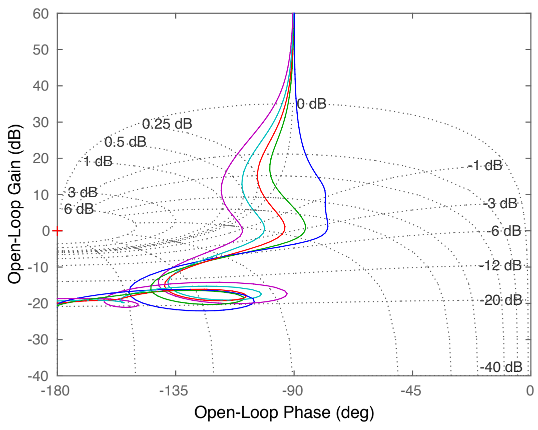
(a) Nyquist diagram of flight group 17 with a *T1* controller, together with an M-circle exclusion region.



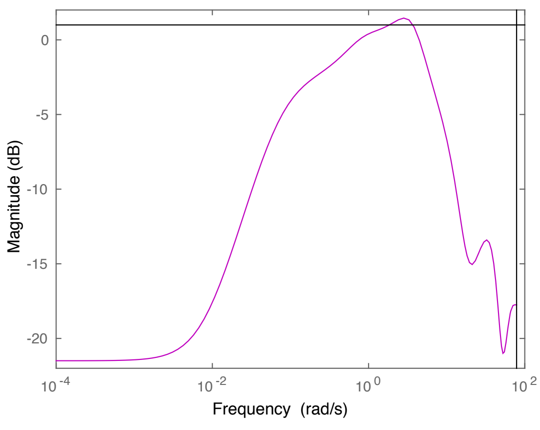
(b) Nyquist diagram of flight group 17 with a *T2* controller, together with an M-circle exclusion region.



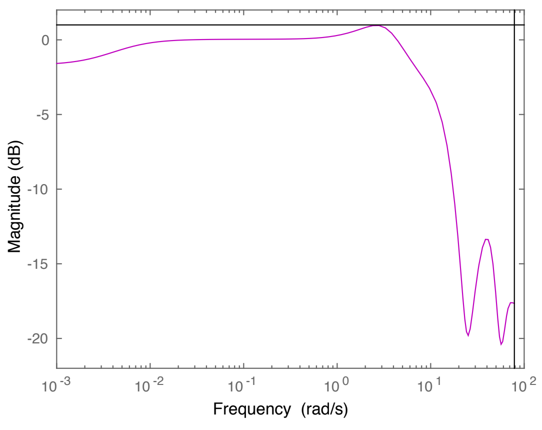
(c) Nichols diagram of flight group 17 with a *T1* controller.



(d) Nichols diagram of flight group 17 with a *T2* controller.

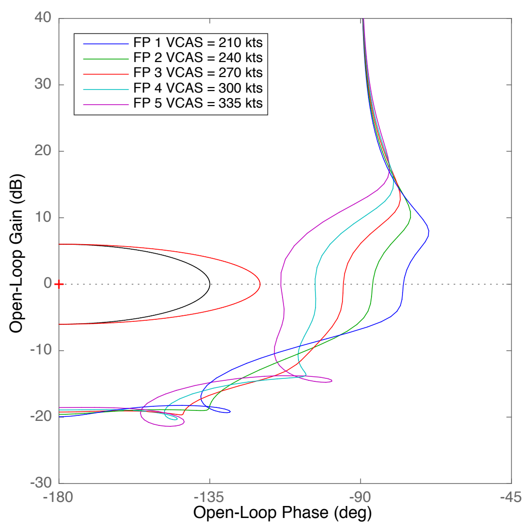


(e) Bode diagram of a complementary sensitivity function of flight point 83 with a *T1* controller. Horizontal line repesents an upper bound of 1 dB.

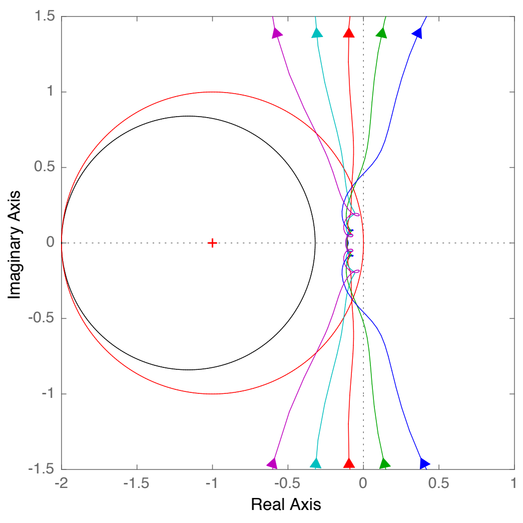


(f) Bode diagram of a complementary sensitivity function of flight point 83 with a *T2* controller. Horizontal line repesents an upper bound of 1 dB.

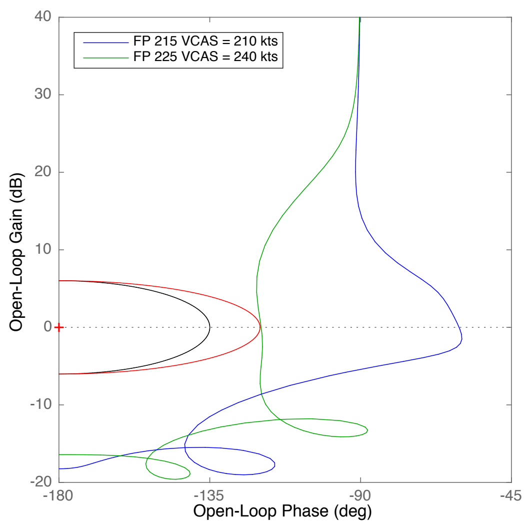
Fig. 3.25 M-circle $\left(M = 10^{0.05}\right)$ analysis for flight group 17.



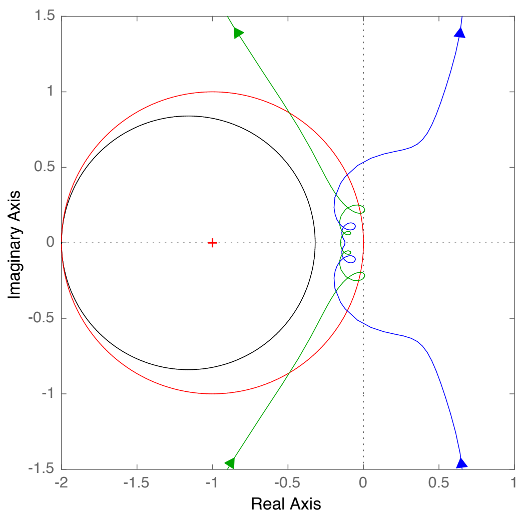
(a) Nichols diagram of flight group 1 with a *T1* controller, together with elliptical exclusion regions.



(b) Nyquist diagram of flight group 1 with a *T1* controller, together with overbounding circular exclusion regions.



(c) Nichols diagram of flight group 46 with a *T2* controller, together with elliptical exclusion regions.



(d) Nyquist diagram of flight group 46 with a *T2* controller, together with overbounding circular exclusion regions.

Fig. 3.26 Simultaneous gain and phase margin analysis for flight group 1 (figures *(a)* and *(b)*) with a *T1* controller and flight group 46 (figures *(c)* and *(d)*) with a *T2* controller. Black exclusion regions correspond to simultaneous gain and phase margins of $G_m = 6$ dB and $P_m = 45°$, while red ones correspond to $G_m = 6$ dB and $P_m = 60°$.

property at all frequencies on the real line, and not only at a discrete set of sampled frequencies.

Additionally, using a quantifier elimination algorithm automates the verification process by removing the need for repeated visual inspection of Nichols and Nyquist plots. In general, visual inspection is difficult and unreliable when the frequency response of the open loop transfer function passes close to an exclusion region (see Figure 3.25 (b) and Figure 3.25 (d)). In such cases, with classical methods, trust must be put in a plotting algorithm that operates on a discrete set of values, and therefore may produce an incorrect result when the exclusion region and the frequency response of $L(s)$ get close. The quantifier elimination based approach does not experience such issues because it works by algebraically manipulating the mathematical expression representing the verification criterion and does not rely on any numerical techniques.

3.6.3 Clearance of verification criteria throughout the whole flight envelope

In the previous section, we demonstrated that the quantifier elimination based approach can be used to certify certain properties of a transfer function for a given linearisation of the aircraft short period dynamics. We now develop a verification framework to clear various criteria of interest throughout the whole flight envelope for the backup flight control law described in Section 3.6.1.

In order to achieve this, we need to obtain controller gains K and system dynamics at flight groups other than the ones provided by Airbus for the RECONFIGURE project. Since the provided grid of flight points (and, consequently, flight groups) is not completely regular, "fictitious" flight groups are used to "pad" the flight envelope to allow it to be bounded by simple box constraints, as shown in Figure 3.27 (a). This is done by repeating the data for the nearest defined flight point before performing the Delaunay Triangulation explained in Section 3.6.1. Additionally, by re-sampling the interpolation, we can partition the padded flight envelope into cuboids, as shown in Figure 3.27 (b). For more details on this, see Section 2.4 on implementation aspects in Maciejowski et al. (2016).

Hence, by performing the discussed procedure, we obtain static control gains $K\left(z_i, M_j, x_k\right)$ for 105 combinations of mass (in tonnes) $M \in \{260,\ 320,\ 375,\ 405,\ 410,\ 550,\ 560\}$, centre of gravity (in percent) $x \in \{28,\ 36,\ 43\}$ and standard pressure altitude (in feet) $z \in \{5000,\ 12500,\ 20000,\ 27500,\ 35000\}$. Therefore, the interpolated control gain $K(z, M, x)$ as a function of altitude z, mass M and centre of gravity x is obtained for the cuboid of the flight envelope

$$\begin{aligned}
z_i \le z \le z_{i+1};\ z_i \in \{5000,\ 12500,\ 20000,\ 27500,\ 35000\},\ i = 1,\ldots,4,\\
M_j \le M \le M_{j+1};\ M_j \in \{260,\ 320,\ 375,\ 405,\ 410,\ 550,\ 560\},\ j = 1,\ldots,6,\\
x_k \le x \le x_{k+1};\ x_k \in \{28,\ 36,\ 43\},\ k = 1,\ 2,
\end{aligned}$$

via linear interpolation (see Figure 3.27 (c))

$$K(z,M,x) = K(z,M,x_k) + \frac{x - x_k}{x_{k+1} - x_k}\left(K(z,M,x_{k+1}) - K(z,M,x_k)\right), \tag{3.171}$$

where:

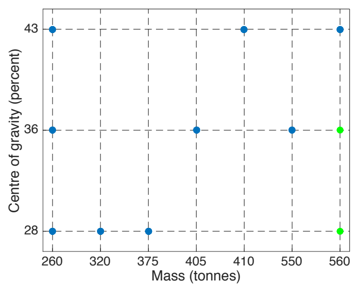
$$\begin{aligned}
K(z,M,x_k) &= K\left(z,M_j,x_k\right) + \frac{M - M_j}{M_{j+1} - M_j}\left(K\left(z,M_{j+1},x_k\right) - K\left(z,M_j,x_k\right)\right),\\
K\left(z,M_j,x_k\right) &= K\left(z_i,M_j,x_k\right) + \frac{z - z_i}{z_{i+1} - z_i}\left(K\left(z_{i+1},M_j,x_k\right) - K\left(z_i,M_j,x_k\right)\right),\\
K\left(z,M_{j+1},x_k\right) &= K\left(z_i,M_{j+1},x_k\right) + \frac{z - z_i}{z_{i+1} - z_i}\left(K\left(z_{i+1},M_{j+1},x_k\right) - K\left(z_i,M_{j+1},x_k\right)\right),\\
K\left(z_i,M_j,x_k\right) &- \text{static control gain at altitude } z_i,\ \text{mass } M_j \text{ and centre of gravity } x_k.
\end{aligned}$$

In order to describe longitudinal short-period dynamics of the aircraft throughout the whole flight envelope, we will use a polynomial surrogate model based on the provided set of linearisations. This model was developed in Hartley (2015).

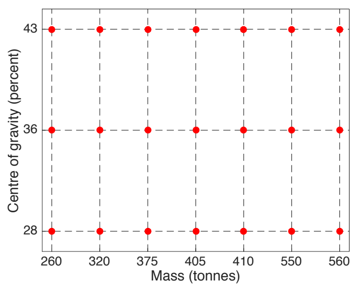
Once the control gain $K(z,M,x)$ and the polynomial surrogate model of longitudinal short-period dynamics are obtained, they are used to translate clearance criteria discussed at the end of the Section 3.6.1 to a polynomial positivity problem over frequency ω and flight envelope parameters M,x,z

$$\forall\ \omega,M,x,z:\ (m_l \le M \le m_u) \wedge (x_l \le x \le x_u) \wedge (z_l \le z \le z_u) \implies f(\omega,M,x,z) > 0, \tag{3.172}$$

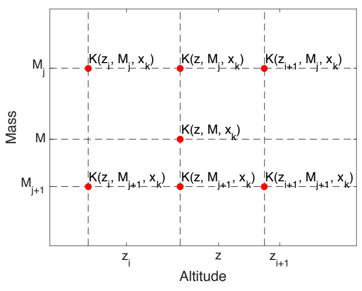
where $f(\omega,M,x,z)$ is a polynomial. All the symbolic manipulations required to obtain the mathematical expression (3.172) are performed using *MATLAB* Symbolic Math Toolbox (The MathWorks Inc., 2012c). This formula is then fed as an input to a combination of quantifier elimination algorithms (cylindrical algebraic decomposition (see Section 2.3.3) and Weispfenning's virtual term substitution procedure (see Section 2.3.2)) implemented in *Mathematica* (Wolfram Research Inc., 2016) which, in turn, outputs an equiv-



(a) Padded flight envelope.



(b) Re-sampled flight envelope.



(c) Interpolation of the control gain.

Fig. 3.27 In figure (a), blue dots represent the irregular grid of flight groups provided by Airbus while green dots represent the "fictitious" flight groups used to "pad" the flight envelope. Figure (b) shows the partition of the padded flight envelope into regular cuboids. Both figures (a) and (b) represent the data at altitudes of 5000, 12500, 20000, 27500 and 35000 feet — hence, the flight envelope is split into 48 rectangular cuboids. In figure (c), it is shown how the control gain $K(z, M, x_k)$ is interpolated as a function of altitude z and mass M for a particular centre of gravity value x_k.

alent quantifier-free formula. Since all variables in (3.172) are quantified, quantifier elimination algorithms become decision procedures with outputs $True$ or $False$.

If the algorithm outputs $True$, it means that the verification criterion is cleared throughout the whole flight envelope, and we are done. Otherwise, by using bisection, we split the region over which the clearance is being done, $(m_l \leq M \leq m_u) \wedge (x_l \leq x \leq x_u) \wedge (z_l \leq z \leq z_u)$, into eight smaller cuboids and perform verification by quantifier elimination on each one of them. We repeat this verify-and-split process until we reach the cuboid to be checked whose size is smaller than the predetermined smallest allowable one (which is chosen to be $|m_u - m_l| \leq 1$ tonne, $|x_u - x_l| \leq 0.5\%$, $|z_u - z_l| \leq 100$ feet). If the outcome of the quantifier elimination procedure for the smallest allowable cuboid is $True$, then the property of interest holds in this particular part of the flight envelope. Otherwise, if the outcome is $False$, then the verification criterion is not cleared in this part of the flight envelope.

The described verification framework is depicted as a flowchart in Figure 3.28 on page 124.

Simultaneous gain and phase margins

In this section, we attempt to clear simultaneous gain and phase margin criteria with gain margin $G_m = 6$ dB and phase margin $P_m = 45°$, as described in Section 3.2.2, which results in an overbounding circle in the Nyquist plane with centre $(-x_c, 0) = (-1.16,\ 0)$ and radius

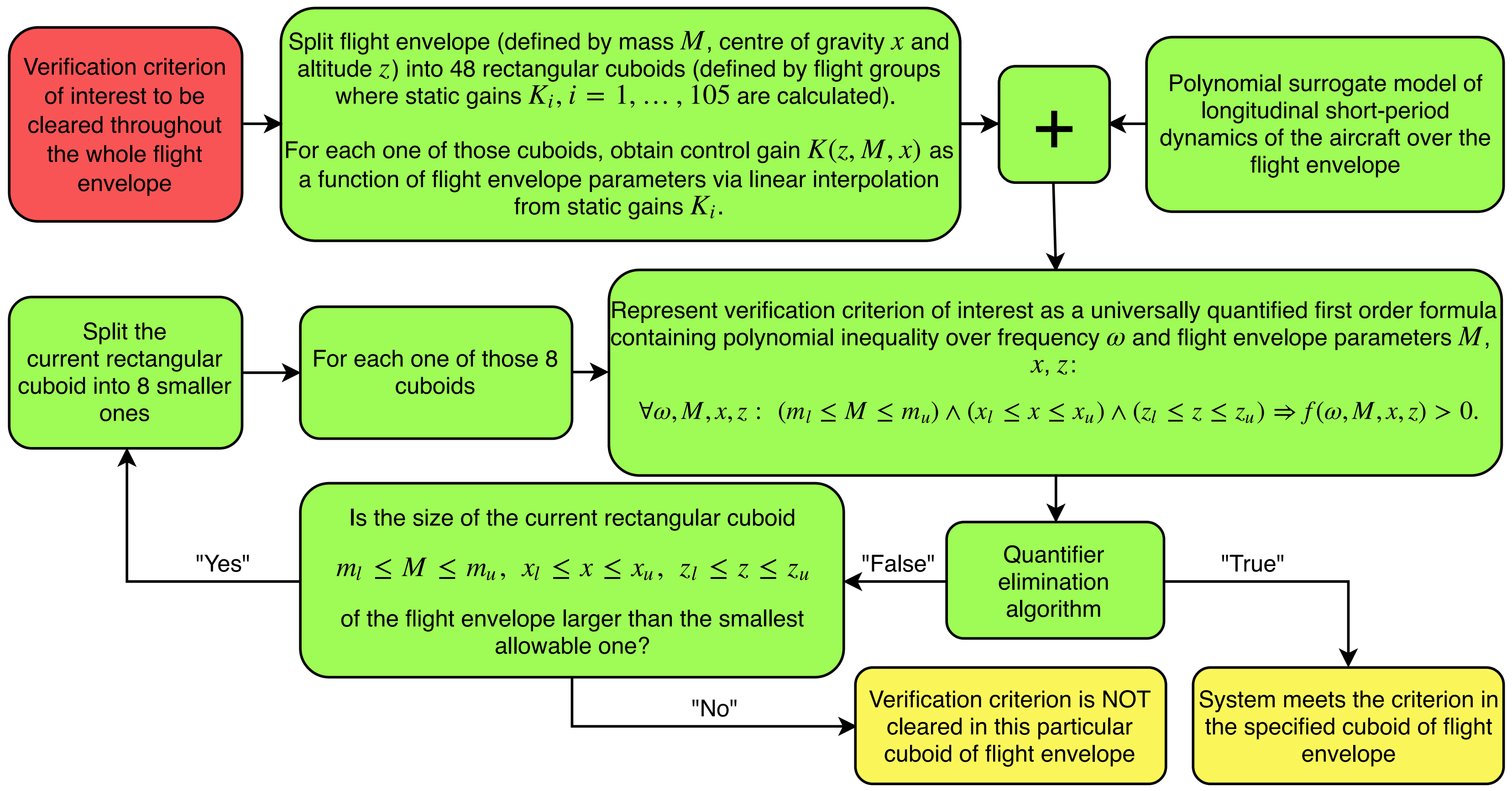

Verification criterion of interest to be cleared throughout the whole flight envelope
Split flight envelope (defined by mass M, centre of gravity x and altitude z) into 48 rectangular cuboids (defined by flight groups where static gains K_i, $i = 1, \dots, 105$ are calculated).
For each one of those cuboids, obtain control gain $K(z, M, x)$ as a function of flight envelope parameters via linear interpolation from static gains K_i.
+
Polynomial surrogate model of longitudinal short-period dynamics of the aircraft over the flight envelope
Split the current rectangular cuboid into 8 smaller ones
For each one of those 8 cuboids
Represent verification criterion of interest as a universally quantified first order formula containing polynomial inequality over frequency ω and flight envelope parameters M, x, z:
$\forall \omega, M, x, z : \ (m_l \leq M \leq m_u) \wedge (x_l \leq x \leq x_u) \wedge (z_l \leq z \leq z_u) \Rightarrow f(\omega, M, x, z) > 0.$
Is the size of the current rectangular cuboid
$m_l \leq M \leq m_u, \ x_l \leq x \leq x_u, \ z_l \leq z \leq z_u$
of the flight envelope larger than the smallest allowable one?
"Yes"
"False"
Quantifier elimination algorithm
"True"
"No"
Verification criterion is NOT cleared in this particular cuboid of flight envelope
System meets the criterion in the specified cuboid of flight envelope

Fig. 3.28 Framework for clearing verification criteria throughout the whole flight envelope. Red blocks represent actions required by a human user, green blocks indicate automated steps and yellow blocks show the outcome of verification.

$r = 0.84$. This region now has to be avoided by the open loop transfer function $L(s,M,x,z)$ for all allowable values of mass M, centre of gravity x and altitude z in the flight envelope. Assuming that

$$L(j\omega,M,x,z) = \frac{a(\omega,M,x,z) + i \cdot b(\omega,M,x,z)}{c(\omega,M,x,z)}, \tag{3.173}$$

where a,b and c are real functions of frequency ω and flight envelope parameters, the input to the quantifier elimination algorithm is the following fully-quantified first order formula:

$$\begin{aligned} &\forall\, \omega,M,x,z:\ (260 \le M \le 560) \wedge (28 \le x \le 43) \wedge (5000 \le z \le 35000) \implies \\ &\qquad ((a(\omega,M,x,z) + x_c c(\omega,M,x,z))^2 + b(\omega,M,x,z)^2 - r^2 c(\omega,M,x,z)^2 > 0. \end{aligned} \tag{3.174}$$

For the control law of *Type 2*, the output of the quantifier elimination algorithm is *True*, which means that the simultaneous gain and phase margin criterion is met throughout the whole flight envelope. This verification criterion was not checked for the control law of *Type 1*.

$\mathcal{H}_\infty$-norm bounds on transfer functions from the demanded load factor to vertical load factor and pitch rate

Another performance criterion of interest is to bound the $\mathcal{H}_\infty$-norms of transfer functions from the demanded load factor n_{zref} to vertical load factor n_z and pitch rate q. For the transfer function from n_{zref} to n_z, $T_{nzref\to n_z}$, we choose its $\mathcal{H}_\infty$-norm to be bounded by $d = 1.1$:

$$\begin{aligned} &\forall\, \omega,M,x,z:\ (260 \le M \le 560) \wedge (28 \le x \le 43) \wedge (5000 \le z \le 35000) \implies \\ &\qquad \left|\left|T_{nzref\to n_z}(j\omega,M,x,z)\right|\right|_\infty \le d = 1.1. \end{aligned} \tag{3.175}$$

Similarly, for the transfer function from n_{zref} to q, $T_{nzref\to q}$, we choose its $\mathcal{H}_\infty$-norm to be bounded as

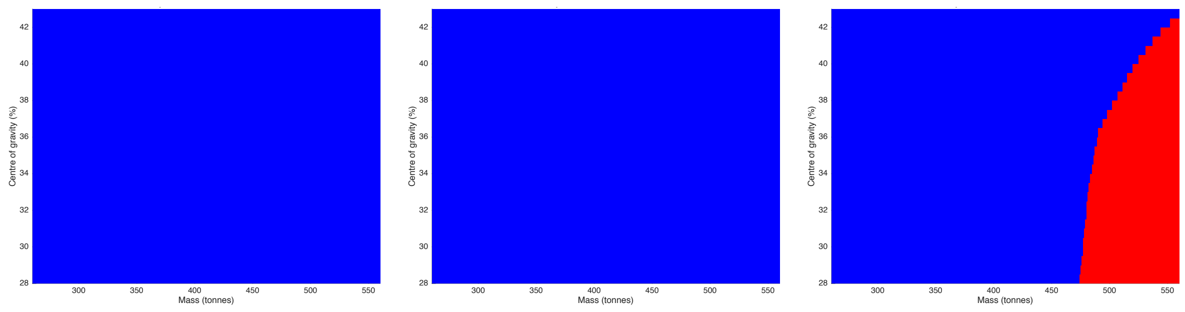
$$\begin{aligned} &\forall\, \omega,M,x,z:\ (260 \le M \le 560) \wedge (28 \le x \le 43) \wedge (5000 \le z \le 35000) \implies \\ &\qquad \left|\left|T_{nzref\to q}(j\omega,M,x,z)\right|\right|_\infty \le d = 1.3\frac{180}{\pi}\frac{g}{V_{TAS}}, \end{aligned} \tag{3.176}$$

where V_{TAS} denotes true airspeed.

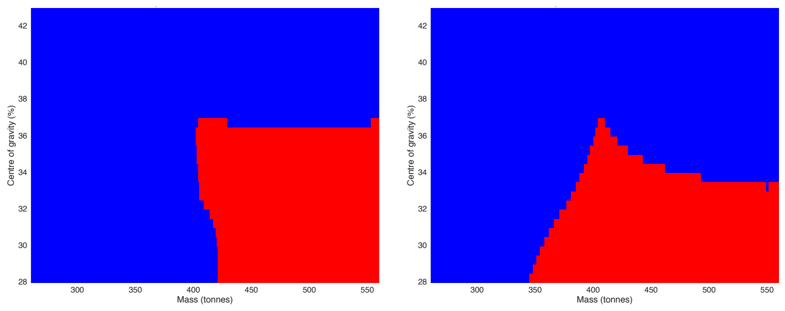
Assuming that both $T_{nzref\to n_z}(j\omega,M,x,z)$ and $T_{nzref\to q}(j\omega,M,x,z)$ are of the form (3.173), the input to the quantifier elimination algorithm for both verification criteria is the following fully-quantified first order formula:

$$\forall\, \omega, M, x, z : \; (260 \leq M \leq 560) \wedge (28 \leq x \leq 43) \wedge (5000 \leq z \leq 35000) \implies \qquad (3.177)$$
$$a(\omega, M, x, z)^2 + b(\omega, M, x, z)^2 - d^2 c(\omega, M, x, z)^2 \leq 0.$$

For the transfer function from the demanded load factor n_{zref} to vertical load factor n_z with the control law of *Type 2*, output of the quantifier elimination algorithm is $True$. This means that the $\mathscr{H}_\infty$-norm bound $\left|\left|T_{nzref \to n_z}(j\omega, M, x, z)\right|\right|_\infty \leq 1.1$ is met throughout the whole flight envelope. Clearance results for the transfer function from the demanded load factor n_{zref} to pitch rate q with the control law of *Type 2* are depicted in Figure 3.29. Neither of verification criteria were checked with the control law of *Type 1*.



(a) $z = 5000$ feet. (b) $z = 12500$ feet. (c) $z = 20000$ feet.



(d) $z = 27500$ feet. (e) $z = 35000$ feet.

Fig. 3.29 Clearance of the $\mathscr{H}_\infty$-norm bound for the transfer function from the demanded load factor n_{zref} to pitch rate q at various altitudes z. Blue regions represent parts of the flight envelope where the bound is met, while red ones denote sections where this verification criterion is not cleared.

In conclusion, results presented in this section show that, for some particular problems, quantifier-elimination-based verification frameworks can be applied to real-world industrial systems. In particular, clearance of verification criteria throughout the whole flight envelope was computationally tractable by a quantifier-elimination-based verification framework because the verification problem in question depended on a small number of parameters (mass, centre of gravity, altitude and frequency), and therefore a CAD-based quantifier elimination algorithm could be utilised.

Chapter 4

Summary

In this thesis, we approached the verification of control systems by using formal methods. This approach consisted of choosing verification criteria of interest and then translating it to the form that Satisfiability Modulo Theory (SMT) solvers or quantifier elimination (QE) algorithms are (in principle) capable of verifying.

We initially started by investigating the idea of expressing the system of interest and the associated verification requirement graphically in terms of a *Simulink* diagram which then would get translated to an equivalent *Why3* theory that would be part of a first order logic (as discussed in Araiza-Illan et al. (2014)). Then this theory would get proven/disproven by one of the SMT solvers interfaced with *Why3*. This type of verification framework would have been a very powerful tool to clear verification requirements for the control schemes implemented graphically, such as the backup flight control law developed in Maciejowski et al. (2016) for the RECONFIGURE project (Goupil et al., 2014). We managed to use this approach to prove Lyapunov stability of an autonomous system with one or two states. Unfortunately, in particular cases, trying to apply this to a linear system with two states resulted in computational issues (running out of time at the SMT solver stage) because of the introduction of many variables (signals in the *Simulink* diagram) that unnecessarily increased the complexity of the corresponding *Why3* theory, and consequently, of the verification requirement. Hence, while in principle this verification approach could have been used to clear verification criteria for more complicated systems such as the one presented in Maciejowski et al. (2016), it would not have been effective because of of the immense amount of time required by the SMT solvers.

Hence, this motivated another formal-methods-based approach that depended on expressing a verification requirement as a quantified mathematical formula. In a nutshell, this verification framework worked by feeding the quantified formula as an input to one of the quantifier elimination algorithms that produce an equivalent quantifier-free mathematical expression in unquantified parameters. Compared to the *Why3* verification framework, this quantifier elimination approach had the benefit of not requiring a large number of variables to describe a verification specification in an appropriate form.

We took advantage of two QE algorithms: Weispfenning's virtual term substitution and quantifier elimination by cylindrical algebraic decomposition (CAD). The CAD-based algorithm is a general quantifier elimination algorithm that, in principle, is applicable to any input formula, regardless of its quantification structure. Weispfenning's virtual term substitution algorithm is a specialised procedure that is applicable as long as all quantified variables are linear (except (possibly) one) in the input formula. The generality of the CAD-based QE algorithm allowed us, in principle, to consider problems involving non-convex optimisation or computation of an exact mathematical expression for the structured singular value μ for a general system. In practice, the complexity of the systems we could analyse by using the CAD-based algorithm was fairly limited — trying to eliminate quantifiers using CAD-based algorithm from input formulas depending on more than four variables turned out to be intractable. This was the case because the size of the corresponding cylindrical algebraic decomposition grows doubly exponentially with the number of variables (both quantified and unquantified) in the input formula. Consequently, we mostly focused our attention to control analysis and synthesis problems that can be expressed as quantifier elimination problems with a quantification structure amenable to Weispfenning's algorithm. Subsequently, this allowed us to take advantage of its lower worst-case running time which does not depend on the number of unquantified variables in the input formula.

Initially, we showed how the avoidance of various exclusion regions by the open loop transfer function could be expressed as a simple quantifier elimination problem. It was later used to verify several performance and robustness specifications of the backup flight control scheme developed for the RECONFIGURE project for a real world industrial system in the aerospace field. This non-academic verification problem was computationally tractable by the CAD-based QE algorithm because it depended on a small number of parameters (four in this particular case: mass, centre of gravity, altitude and frequency). This application example illustrated a well known fact — the main bottleneck of using the CAD-based QE algorithm to verify properties of industrial systems is the number of parameters on which the system depends. Hence, more widespread use of this verification approach depends on improvements in the efficiency and running time of the underlying quantifier elimination algorithm and on more people being aware of the algorithm and what it can do.

Additionally, we used the quantifier-elimination-based verification procedure to analyse the robust stability of uncertain systems via calculation of the structured singular value μ. Attempting to compute it for a system under a general norm-bounded uncertainty (even in the simple case of two norm-bounded uncertainties) resulted in a computationally intractable problem because its quantification structure required us to use the computationally expensive CAD-based QE algorithm. Therefore, in order to reduce the computational burden from the quantifier elimination algorithm perspective, we considered a system under structured parametric uncertainty. This then resulted in the problem with a quantification structure that is amenable to a more efficient Weispfenning's QE procedure. Subsequently, this allowed us to calculate the structured singular value μ for systems subject to three or four real parametric uncertainties. The main benefit of using the quantifier elimination based approach as opposed to the standard branch and bound algorithms is that quantifier elimination method provides the exact value of μ rather than bounding it from below and above to an arbitrary accuracy.

Similarly, generality of the CAD-based QE algorithm could have theoretically allowed us to obtain an an explicit MPC law for a general nonlinear time invariant system with a polynomial objective and polynomial constraints. Again, the types of problems we could consider in practice were restricted by the computational complexity of the underlying quantifier elimination problem. Therefore, we restricted our focus to MPC problems with a quadratic objective and polytopic constraints and time invariant systems that resulted in an equivalent quantifier elimination problem with a quantification structure amenable to Weispfenning's QE algorithm.

Finally, we showed how verification requirements for system dynamics expressed in general Linear Temporal Logic (LTL) specifications could be translated to quantifier elimination problems. This then allowed us to calculate feasible parameter sets in cases where standard approaches in the LTL literature based on reachability analysis failed, including nonlinearly parametrized systems or LTL specifications that result in non-convex feasible parameter sets. Synthesis of feasible parameter sets was performed using both CAD-based and Weispfenning's quantifier elimination algorithms. These computations illustrated the general rule of thumb that, as long as the quantification structure of the problem is such that Weispfenning's algorithm is applicable, it is more computationally efficient to use it rather than the general CAD-based QE-algorithm.

In conclusion, while quantifier-elimination-based verification approaches have several important benefits, such as their generality and guarantee not to miss a critical frequency or parameter combination at which the property of interest is violated, these approaches usually suffer from a large computational penalty compared to numerical methods. This limited the complexity of systems that could be verified using these methods. Therefore, a more widespread application of the proposed verification scheme depends on the improvements in running time and applicability of the underlying quantifier elimination algorithms.

In particular, the bottleneck of the CAD-based QE algorithm is its projection phase during which the projection operator tends to produce sets with a large number of polynomials. Since the introduction of the CAD-based QE algorithm, many improvements have been suggested in order to reduce the size of the sets produced by the projection operator, and consequently improve the efficiency of the algorithm — for more details, see Hong (1990), McCallum (1988), McCallum (1998), Brown (2001). Hence, applicability of the verification framework that relies on the CAD-based QE algorithm to problems with a larger number of quantified and unquantified variables depends on the future improvement in this area.

Moreover, the extension of Weispfenning's virtual term substitution algorithm to cases beyond quadratically or cubically quantified variables depends on the development of the theory of the representation of real roots. In principle, Weispfenning's algorithm could be extended to formulas in which the quantified variable appears with an unbounded degree, by exploiting Thom's Lemma for representation of real roots — for the state of the art, see Kosta and Sturm (2015); Liiva et al. (2014). Unfortunately, no efficient implementation of the algorithm for these cases exists yet.

References

A. Floudas, C., Pardalos, P., S. Adjiman, C., R. Esposito, W., Gumus, Z., T. Harding, S., Klepeis, J., Meyer, C., and Schweiger, C. (1999). *Handbook of Test Problems in Local and Global Optimization*. http://titan.princeton.edu/TestProblems/.

Araiza-Illan, D., Eder, K., and Richards, A. (2014). Formal verification of control systems properties with theorem proving. https://arxiv.org/abs/1405.7615.

Arnon, D. S., Collins, G. E., and McCallum, S. (1984). Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877.

Baier, C., Katoen, J.-P., and Larsen, K. G. (2008). *Principles of model checking*. MIT press.

Balakrishnan, V., Boyd, S., and Balemi, S. (1991). Branch and bound algorithm for computing the minimum stability degree of parameter-dependent linear systems. *International Journal of Robust and Nonlinear Control*, 1(4):295–317.

Bazaraa, M., Sherali, H., and Shetty, C. (1993). *Nonlinear programming: theory and algorithms*. Wiley, NY.

Bemporad, A., Fukuda, K., and Torrisi, F. D. (2001). Convexity recognition of the union of polyhedra. *Computational Geometry*, 18(3):141–154.

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20.

Bobot, F., Filliâtre, J.-C., Marché, C., and Paskevich, A. (2011). Why3: Shepherd your herd of provers. In *Proceedings of Boogie 2011, the 1st International Workshop on Intermediate Language Verification*, pages 53–64.

Braatz, R. P., Young, P. M., Doyle, J. C., and Morari, M. (1994). Computational complexity of μ calculation. *IEEE Transactions on Automatic Control*, 39(5):1000–1002.

Brown, C. W. (2001). Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465.

Brown, C. W. (2003). QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM Bull.*, 37(4):97–108.

Chang, B. C., Yeh, H., Banda, S., and Ekdal, O. (1991). Computation of the real structured singular value via polytopic polynomials. *Journal of Guidance, Control, and Dynamics*, 14(1):140–147.

Collins, G. E. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decompostion. *Springer Lecture Notes in Computer Science*, 33:515–532.

Collins, G. E. and Hong, H. (1991). Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328.

Cuzzola, F. A., Geromel, J. C., and Morari, M. (2002). An improved approach for constrained robust Model Predictive Control. *Automatica*, 38(7):1183–1189.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

de Gaston, R. R. E. and Safonov, M. G. (1988). Exact calculation of the multiloop stability margin. *IEEE Transactions on Automatic Control*, 33(2):156–171.

de Moura, L. and Bjørner, N. (2008). Z3: An efficient SMT solver. In Ramakrishnan, C. R. and Rehof, J., editors, *Proceeding of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340.

de Oliveira, M. C., Bernussou, J., and Geromel, J. C. (1999). A new discrete-time robust stability condition. *Systems & Control Letters*, 37(4):261–265.

de Oliveira, M. C., Geromel, J. C., and Bernussou, J. (2002). Extended H_2 and H_∞ norm characterizations and controller parametrizations for discrete-time systems. *International Journal of Control*, 75(9):666–679.

Deodhare, G. and Patel, V. V. (1998). A "modern" look at gain and phase margins: an H_∞/μ approach. In *Proceedings of the AIAA Conference on Guidance, Navigation and Control*, pages 325–335.

Dorobantu, A., Balas, G. J., and Georgiou, T. T. (2014). Validating aircraft models in the gap metric. *Journal of Aircraft*, 51(6):1665–1672.

Doyle, J. (1982). Analysis of feedback systems with structured uncertainties. In *IEE Proceedings D-Control Theory and Applications*, volume 129, pages 242–250.

Dutertre, B. and de Moura, L. (2006). A fast linear-arithmetic solver for DPLL(T). In *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*, pages 81–94.

Goupil, P., Boada-Bauxell, J., Marcos, A., Cortet, E., Kerr, M., and Costa, H. (2014). AIRBUS efforts towards advanced real-time fault diagnosis and fault tolerant control. *IFAC Proceedings Volumes*, 47(3):3471–3476.

Haesaert, S., van den Hof, P., and Abate, A. (2017). Data-driven and model-based verification via Bayesian identification and reachability analysis. *Automatica*, 79:115–126.

Hartley, E. (2015). Predictive control with parameter adaptation to achieve α-protection in the RECONFIGURE benchmark in the presence of icing. *IFAC-PapersOnLine*, 48(23):172–177.

Herceg, M., Kvasnica, M., Jones, C., and Morari, M. (2013). Multi-Parametric Toolbox 3.0. In *Proceedings of the European Control Conference*, pages 502–510. http://control.ee.ethz.ch/~mpt.

Hong, H. (1990). An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '90, pages 261–264.

Hurd, J. (2003). First-order proof tactics in higher-order logic theorem provers. In Archer, M., Vito, B. D., and Muñoz, C., editors, *Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports*, pages 56–68.

Hurd, J. (2007). Metis first order prover. http://gilith.com/software/metis/.

Jordan, D. and Smith, P. (2007). *Nonlinear Ordinary Differential Equations: An Introduction for Scientists and Engineers*. Oxford Texts in Applied and Engineering Mathematics, OUP Oxford.

Kosta, M. and Sturm, T. (2015). A generalized framework for virtual substitution. http://arxiv.org/abs/1501.05826.

Levine, W. S. (1996). *The Control Handbook*. CRC Press/IEEE Press, Piscataway, NJ.

Liiva, K., Passmore, G. O., and Jackson, P. B. (2014). A note on real quantifier elimination by virtual term substitution of unbounded degree. In *Proceedings of the PAS workshop*, Vienna Summer of Logic.

Loos, R. and Weispfenning, V. (1993). Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462.

Löfberg, J. (2012). Oops! I cannot do it again: Testing for recursive feasibility in MPC. *Automatica*, 48(3):550–555.

Maciejowski, J., Hartley, E., and Siaulys, K. (2016). A longitudinal flight control law to accommodate sensor loss in the RECONFIGURE benchmark. *Annual Reviews in Control*, 42:212–223.

McCallum, S. (1988). An improved projection operation for cylindrical algebraic decomposition of three-dimensional space. *Journal of Symbolic Computation*, 5(1-2):141–161.

McCallum, S. (1998). An improved projection operation for cylindrical algebraic decomposition. In Caviness, B. F. and Johnson, J. R., editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 242–268.

Mishra, B. (1993). *Algorithmic Algebra*. Springer-Verlag, NY.

Newlin, M. P. and Young, P. M. (1997). Mixed μ problems and branch and bound techniques. *International Journal of Robust and Nonlinear Control*, 7(2):145–164.

Nieuwenhuis, R. and Oliveras, A. (2007). Fast congruence closure and extensions. *Information and Computation*, 205(4):557–580.

Paulson, L. (2012). *MetiTarski*: Past and future. In Beringer, L. and Felty, A., editors, *Interactive Theorem Proving*, volume 7406 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin Heidelberg.

Reichert, R. T. (1992). Dynamic scheduling of modern-robust-control autopilot designs for missiles. *IEEE Control Systems*, 12(5):35–42.

Routh, E. (1877). *A Treatise on the Stability of a Given State of Motion: Particularly Steady Motion*. Macmillan and Company.

Siaulys, K. and Maciejowski, J. M. (2016). Verification of model predictive control laws using Weispfenning's quantifier elimination by virtual substitution algorithm. In *Proceedings of the 55th IEEE Conference on Decision and Control*, pages 1452–1457.

Sideris, A. and de Gaston, R. R. E. (1986). Multivariable stability margin calculation with uncertain correlated parameters. In *Proceedings of the 25th IEEE Conference on Decision and Control*, pages 766–771.

Sideris, A. and Sanchez Pena, R. (1989). Fast computation of the multivariable stability margin for real interrelated uncertain parameters. *IEEE Transactions on Automatic Control*, 34:1272–1276.

The MathWorks Inc. (2012a). MATLAB and Robust Control Toolbox Release 2012b. https://www.mathworks.com/products/robust.html.

The MathWorks Inc. (2012b). MATLAB and Simulink Release 2012b. https://uk.mathworks.com/products/simulink.html.

The MathWorks Inc. (2012c). MATLAB and Symbolic Math Toolbox Release 2012b. https://uk.mathworks.com/products/symbolic.html.

Weispfenning, V. (1994). Quantifier elimination for real algebra — the cubic case. In von zur Gathen, J. and Giesbrecht, M., editors, *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '94, pages 258–263.

Weispfenning, V. (1997). Quantifier elimination for real algebra — the quadratic case and beyond. *Applicable Algebra in Engineering, Communication and Computing*, 8(2):85–101.

Wolfram Research Inc. (2016). Mathematica, version 10.4. https://www.wolfram.com/mathematica/.

Young, P. M., Newlin, M. P., and Doyle, J. C. (1995). Computing bounds for the mixed μ problem. *International Journal of Robust and Nonlinear Control*, 5(6):573–590.

Zadeh, L. A. and Desoer, C. A. (1963). *Linear System Theory: the State Space Approach*. McGraw-Hill, New York.

Zhou, K. and Doyle, J. C. (1998). *Essentials of Robust Control*. Prentice-Hall, Upper Saddle River, NJ.

Appendix A

A.1 *Why3* standard library theory for real numbers

In this section, a *Why3* standard library theory for real numbers `RealInfix` (together with other *Why3* standard library theories it depends on) is provided for illustration purposes (for more details, see http://why3.lri.fr/stdlib/). `RealInfix` theory is used to argue about real numbers when one wants to use both integer and real binary operators.

```
theory RealInfix
   use import Real
   function (+.) (x:real) (y:real) : real = x + y
   function (-.) (x:real) (y:real) : real = x - y
   function ( *.) (x:real) (y:real) : real = x * y
   function (/.) (x:real) (y:real) : real = x / y
   function (-._) (x:real) : real = - x
   function inv (x:real) : real = Real.inv x

   predicate (<=.) (x:real) (y:real) = x <= y
   predicate (>=.) (x:real) (y:real) = x >= y
   predicate ( <.) (x:real) (y:real) = x < y
   predicate ( >.) (x:real) (y:real) = x > y
end

theory Real
   constant zero : real = 0.0
   constant one  : real = 1.0

   predicate (< ) real real
   predicate (> ) (x y : real) = y < x
   predicate (<=) (x y : real) = x < y \/ x = y
   clone export algebra.OrderedField with type t = real,
            constant zero = zero, constant one = one, predicate (<=) = (<=)
end

theory OrderedField
   clone export Field
   predicate (<=) t t
   predicate (>=) (x y : t) = y <= x
   clone export relations.TotalOrder with type t = t, predicate rel = (<=)
   axiom ZeroLessOne : zero <= one
   axiom CompatOrderAdd  : forall x y z : t. x <= y -> x + z <= y + z
   axiom CompatOrderMult : forall x y z : t. x <= y -> zero <= z -> x * z <= y * z
end

theory Field
   clone export UnitaryCommutativeRing
   function inv t : t
   axiom Inverse : forall x:t. x <> zero -> x * inv x = one
   function (/) (x y : t) : t = x * inv y

   lemma add_div : forall x y z : t. z <> zero -> (x+y)/z = x/z + y/z
   lemma sub_div : forall x y z : t. z <> zero -> (x-y)/z = x/z - y/z
   lemma neg_div : forall x y : t. y <> zero -> (-x)/y = -(x/y)
   lemma assoc_mul_div: forall x y z:t. z <> zero -> (x*y)/z = x*(y/z)
   lemma assoc_div_mul: forall x y z:t. y <> zero /\ z <> zero -> (x/y)/z = x/(y*z)
   lemma assoc_div_div: forall x y z:t. y <> zero /\ z <> zero -> x/(y/z) = (x*z)/y
end

theory UnitaryCommutativeRing
   clone export CommutativeRing
   constant one : t
   axiom Unitary : forall x:t. one * x = x
   axiom NonTrivialRing : zero <> one
end

theory CommutativeRing
   clone export Ring
   clone Comm with type t = t, function op = op
   meta AC function op
end

theory Ring
   type t
   constant zero : t
   function (+) t t : t
   function (-_) t : t
   function (*) t t : t
   clone CommutativeGroup with type t = t, constant unit = zero, function op = (+),
   function inv = (-_)

   clone Assoc with type t = t, function op = (*)
   axiom Mul_distr_l : forall x y z : t. x * (y + z) = x * y + x * z
   axiom Mul_distr_r : forall x y z : t. (y + z) * x = y * x + z * x
   function (-) (x y : t) : t = x + -y
end

theory CommutativeGroup
   clone export Group
   clone Comm with type t = t, function op = op
   meta AC function op
end

theory Group
   clone export Monoid
   function inv t : t
   axiom Inv_def_l : forall x:t. op (inv x) x = unit
   axiom Inv_def_r : forall x:t. op x (inv x) = unit
end

theory Monoid
   clone export Assoc
   constant unit : t
   axiom Unit_def_l : forall x:t. op unit x = x
   axiom Unit_def_r : forall x:t. op x unit = x
end

theory Assoc
   type t
   function op t t : t
   axiom Assoc : forall x y z : t. op (op x y) z = op x (op y z)
end

theory Comm
   type t
   function op t t : t
   axiom Comm : forall x y : t. op x y = op y x
end

theory TotalOrder
   clone export PartialOrder
   clone export Total with type t = t, predicate rel = rel
end

theory PartialOrder
   clone export PreOrder
   clone export Antisymmetric with type t = t, predicate rel = rel
end

theory Total
   clone export EndoRelation
   axiom Total : forall x y:t. rel x y \/ rel y x
end

theory PreOrder
   clone export Reflexive
   clone export Transitive with type t = t, predicate rel = rel
end

theory Antisymmetric
   clone export EndoRelation
   axiom Antisymm : forall x y:t. rel x y -> rel y x -> x = y
end

theory Reflexive
   clone export EndoRelation
   axiom Refl : forall x:t. rel x x
end

theory Transitive
   clone export EndoRelation
   axiom Trans : forall x y z:t. rel x y -> rel y z -> rel x z
end

theory EndoRelation
   type t
   predicate rel t t
end
```

A.2 *Why3* theories for scalar *Simulink* blocks

In this section, a current library of *Why3* theories developed by us for scalar *Simulink* blocks (stored in the file 'Theories.why') is provided.

```
theory Product
   use import int.Int
   use import real.RealInfix

   function in1 int : real
   function in2 int : real
   function out1 int : real

   axiom a1: forall k:int. out1 k = in1 k *. in2 k
   axiom a2: forall k:int. in1 k >. 0.0 /\ in2 k >. 0.0 -> out1 k >. 0.0
   axiom a3: forall k:int. in1 k <. 0.0 /\ in2 k <. 0.0 -> out1 k >. 0.0
end


theory Sum
   use import int.Int
   use import real.RealInfix

   function in1 int : real
   function in2 int : real
   function out1 int : real

   axiom a1: forall k:int. out1 k = in1 k +. in2 k
   axiom a2: forall k:int. in1 k >. 0.0 /\ in2 k >. 0.0 -> out1 k >. 0.0
   axiom a3: forall k:int. in1 k >. 0.0 /\ in2 k <. 0.0 /\ in1 k >. -.in2 k ->
             out1 k >. 0.0
   axiom a4: forall k:int. in1 k <. 0.0 /\ in2 k >. 0.0 /\ in2 k >. -.in1 k ->
             out1 k >. 0.0
end


theory Gain
   use import int.Int
   use import real.RealInfix
   use import real.Abs

   function in1 int : real
   function out1 int : real
   constant gain : real

   axiom a1 : forall k:int. out1 k = in1 k *. gain
   axiom a2 : forall k:int. in1 k >. 0.0 /\ gain >. 0.0 -> out1 k >. 0.0
   axiom a3 : forall k:int. in1 k <. 0.0 /\ gain <. 0.0 -> out1 k >. 0.0
   axiom a4 : forall k:int. in1 k <. 0.0 /\ gain >. 0.0 -> out1 k <. 0.0
   axiom a5 : forall k:int. in1 k >. 0.0 /\ gain <. 0.0 -> out1 k <. 0.0
end


theory UnitDelay
   use import int.Int
   use import real.RealInfix

   function in1 int : real
   function out1 int : real

   axiom a1 : forall k:int. in1 (k+1) = out1 k
end


theory UnitAdvance
   use import int.Int
   use import real.RealInfix

   function in1 int : real
   function out1 int : real

   axiom a1 : forall k:int. out1 (k+1) = in1 k
end


theory Saturation
   use import int.Int
   use import real.RealInfix

   function in1 int : real
   function out1 int : real
   constant lb : real
   constant ub : real

   axiom a1: forall k:int. in1 k >=. lb /\ in1 k <=. ub -> out1 k = in1 k
   axiom a2: forall k:int. in1 k <. lb -> out1 k = lb
   axiom a3: forall k:int. in1 k >. ub -> out1 k = ub
end


theory EqualToZero
   use import int.Int
   use import real.RealInfix
   use import bool.Bool

   function in1 int : real
   function out1 int : bool

   axiom a1: forall k:int. out1 k = True -> in1 k = 0.0
   axiom a2: forall k:int. out1 k = False -> in1 k >. 0.0 \/ in1 k <. 0.0
end


theory NotEqualToZero
   use import int.Int
   use import real.RealInfix
   use import bool.Bool

   function in1 int : real
   function out1 int : bool

   axiom a1: forall k:int. out1 k = True -> in1 k >. 0.0 \/ in1 k <. 0.0
   axiom a2: forall k:int. out1 k = False -> in1 k = 0.0
end


theory GreaterThanZero
   use import int.Int
   use import real.RealInfix
   use import bool.Bool

   function in1 int : real
   function out1 int : bool

   axiom a1: forall k:int. out1 k = True -> in1 k >. 0.0
   axiom a2: forall k:int. out1 k = False -> in1 k <=. 0.0
end
```

```
theory LessThanZero
   use import int.Int
   use import real.RealInfix
   use import bool.Bool

   function in1 int : real
   function out1 int : bool

   axiom a1: forall k:int. out1 k = True -> in1 k <. 0.0
   axiom a2: forall k:int. out1 k = False -> in1 k >=. 0.0
end


theory GreaterOrEqualToZero
   use import int.Int
   use import real.RealInfix
   use import bool.Bool

   function in1 int : real
   function out1 int : bool

   axiom a1: forall k:int. out1 k = True -> in1 k >=. 0.0
   axiom a2: forall k:int. out1 k = False -> in1 k <. 0.0
end


theory LessOrEqualToZero
   use import int.Int
   use import real.RealInfix
   use import bool.Bool

   function in1 int : real
   function out1 int : bool

   axiom a1: forall k:int. out1 k = True -> in1 k <=. 0.0
   axiom a2: forall k:int. out1 k = False -> in1 k >. 0.0
end


theory LogicAnd
```

```
   use import int.Int
   use import bool.Bool

   function in1 int : bool
   function in2 int : bool
   function out1 int : bool

   axiom a1: forall k:int. in1 k = True /\ in2 k = True -> out1 k = True
   axiom a2: forall k:int. in1 k = False /\ in2 k = False -> out1 k = False
   axiom a3: forall k:int. in1 k = False /\ in2 k = True -> out1 k = False
   axiom a4: forall k:int. in1 k = True /\ in2 k = False -> out1 k = False
end


theory LogicOr
   use import int.Int
   use import bool.Bool

   function in1 int : bool
   function in2 int : bool
   function out1 int : bool

   axiom a1: forall k:int. in1 k = True /\ in2 k = True -> out1 k = True
   axiom a2: forall k:int. in1 k = False /\ in2 k = False -> out1 k = False
   axiom a3: forall k:int. in1 k = False /\ in2 k = True -> out1 k = True
   axiom a4: forall k:int. in1 k = True /\ in2 k = False -> out1 k = True
end
```

A.3 *MATLAB* translation function

In this section, a *MATLAB* function which creates a *Why3* theory representing a *Simulink* model is given.

```
function result = translate(simulink_model)

    %Capitalise the first letter - each theory must start with a capital
    %letter in why
    simulink_model  = CapFirstLetter(simulink_model);

    fileID = fopen(strcat(simulink_model, '.why'),'w');
    fprintf(fileID,'theory %s \n\n', simulink_model);
    fprintf(fileID,' use import int.Int\n');
    fprintf(fileID,' use import real.RealInfix\n');
    fprintf(fileID,' use import bool.Bool \n\n');

    %Get the information representing Simulink diagram
    load_system(simulink_model);

    BlockPaths = find_system(simulink_model, 'LookUnderMasks', 'none', 'Type', 'Block')
    BlockTypes = get_param(BlockPaths, 'BlockType');
    BlockNames = get_param(simulink_model, 'Blocks');

    %Capitalise first letter of each of the block names, since these
    %block names are used as names for theories which must start with a
    %capital letter.
    for i =  1:length(BlockNames)
        BlockNames{i} = CapFirstLetter(BlockNames{i});
    end

    NumberOfPorts = get_param(BlockPaths, 'Ports');
    %NumberOfPorts{i}(1) - number of input ports of i-th element
    %NumberOfPorts{i}(2) - number of output ports of i-th element

    NumberOfBlocks = length(BlockPaths);
    NumberOfRequireBlocks = 0;
    %------------------------------------------------------------

    %1) Name all of the output signals in the Simulink diagram (since each
    %signal is connected to input and output ports, all incoming signals
    %will be named as well)
    %2) Add functions for outgoing signals

    for i = 1:NumberOfBlocks
        if ~isRequireBlock(BlockPaths{i})
            %add output signals and functions to all the blocks
            %except the "Require" one
            NumberOfOutputPorts = NumberOfPorts{i}(2);

            % 1 and 2
            for k = 1:NumberOfOutputPorts
                %1) name each of the ports in the Simulink diagram
                p = get_param(BlockPaths{i}, 'PortHandles');
                l = get_param(p.Outport, 'Line');

                if NumberOfOutputPorts==1
                    %one output port
                    SignalName = strcat(lower(BlockNames{i}), '_p', num2str(k));
                    set_param(l, 'Name', SignalName);
                else
                    %more than one output port
                    SignalName = strcat(lower(BlockNames{i}), '_p', num2str(k));
                    set_param(l{k}, 'Name', SignalName);
                end
                %2) for each of the outgoing signals, add a function
                if isCompareToZeroBlock(BlockPaths{i}) || isLogicBlock(BlockPaths{i})
                    %if the block is a comparison to 0 block
                    %OR
                    %logical block
                    function_statement =
                       strcat([' ' 'function ' SignalName ' ' 'int:bool' '\n']);
                else
                    function_statement =
                       strcat([' ' 'function ' SignalName ' ' 'int:real' '\n']);
                end
                fprintf(fileID, function_statement);
            end
        else
            %calculate how many goals there are
            NumberOfRequireBlocks = NumberOfRequireBlocks + 1;
        end
    end

    %3) Create strings for cloning theories
    TheoryStatements = cell(NumberOfBlocks, 1);
    GoalStatements = cell(NumberOfRequireBlocks, 1);
    GoalNumber = 1;

    for i = 1:NumberOfBlocks
        %get the names of input and output signals
        %(which were populated in the previous for loop)
        p = get_param(BlockPaths{i}, 'PortHandles');
        in = get_param(p.Inport, 'Line');
        out = get_param(p.Outport, 'Line');

        switch BlockTypes{i}

            case 'Product'
                TheoryStatements{i} = strcat([' ' 'clone Theories.Product as' '
                'BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in{1}, 'Name') ',
                function in2 = ' get_param(in{2}, 'Name') ', function out1 = ' get_param(out,
                'Name') '\n\n']);

            case 'Sum'
                TheoryStatements{i} = strcat([' ' 'clone Theories.Sum as' ' ' BlockNames{i} '
                ' 'with function in1 =' ' ' get_param(in{1}, 'Name') ', function in2 = '
                get_param(in{2}, 'Name') ', function out1 = ' get_param(out, 'Name')'\n\n']);

            case 'UnitDelay'
                TheoryStatements{i} = strcat([' ' 'clone Theories.UnitDelay as' ' '
                BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in, 'Name') ', function
                out1 = ' get_param(out, 'Name') '\n\n']);

            case 'UnitAdvance'
                TheoryStatements{i} = strcat([' ' 'clone Theories.UnitAdvance as' ' '
                BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in, 'Name') ', function
                out1 = ' get_param(out, 'Name') '\n\n']);
```

```
case 'Gain'
    GainStr = FixToReal(get_param(BlockPaths{i}, 'Gain'));
    TheoryStatements{i} = strcat([' ' 'clone Theories.Gain as' ' ' BlockNames{i}
    ' ' 'with function in1 =' ' ' get_param(in, 'Name') ', function out1 = '
    get_param(out, 'Name') '\n' ' ' 'axiom' ' '  BlockNames{i} '_' 'gain:' ' '
    BlockNames{i} '.gain = ' ' ' GainStr '\n\n'  ]);

case 'Saturate'
    ubStr = FixToReal(get_param(BlockPaths{i}, 'UpperLimit'));
    lbStr = FixToReal(get_param(BlockPaths{i}, 'LowerLimit'));

    TheoryStatements{i} = strcat([' ' 'clone Theories.Saturation as' ' '
    BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in, 'Name') ', function
    out1 = ' get_param(out, 'Name') '\n' ' ' 'axiom' ' '  BlockNames{i} '_' 'ub:'
    ' ' BlockNames{i} '.ub = ' ' ' ubStr '\n' ' ' 'axiom' ' '  BlockNames{i} '_'
    'lb:' ' ' BlockNames{i} '.lb = ' ' ' lbStr '\n\n'  ]);

case 'Logic'
    op = get_param(BlockPaths{i}, 'Operator');
    switch op
        case 'OR'
             TheoryStatements{i} = strcat([' ' 'clone Theories.LogicOr as' ' '
             BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in{1}, 'Name')
             ', function in2 = ' get_param(in{2}, 'Name') ', function out1 = '
             get_param(out, 'Name') '\n\n']);
        case 'AND'
             TheoryStatements{i} = strcat([' ' 'clone Theories.LogicAnd as' ' '
             BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in{1}, 'Name')
             ', function in2 = ' get_param(in{2}, 'Name') ', function out1 = '
             get_param(out, 'Name') '\n\n']);
    end

case 'SubSystem'
    if isCompareToZeroBlock(BlockPaths{i})
        %if the block is a comparison to 0 block
        rel = get_param(BlockPaths{i}, 'relop');

        switch rel

            case '<'
                TheoryStatements{i} = strcat([' ' 'clone Theories.LessThanZero
                as' ' ' BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in,
                'Name') ', function out1 = ' get_param(out, 'Name') '\n\n']);

            case '<='
                TheoryStatements{i} = strcat([' ' 'clone
                Theories.LessOrEqualToZero as' ' ' BlockNames{i} ' ' 'with
                function in1 =' ' ' get_param(in, 'Name') ', function out1 = '
                get_param(out, 'Name') '\n\n']);

            case '>'
                TheoryStatements{i} = strcat([' ' 'clone Theories.GreaterThanZero
                        ' ' ' BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in,
                        'Name') ', function out1 = ' get_param(out, 'Name') '\n\n']);

                    case '>='
                        TheoryStatements{i} = strcat([' ' 'clone
                        Theories.GreaterOrEqualToZero as' ' ' BlockNames{i} ' ' 'with
                        function in1 =' ' ' get_param(in, 'Name') ', function out1 = '
                        get_param(out, 'Name') '\n\n']);

                    case '=='
                        TheoryStatements{i} = strcat([' ' 'clone Theories.EqualToZero as'
                        ' ' BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in,
                        'Name') ', function out1 = ' get_param(out, 'Name') '\n\n']);

                    case '~='
                        TheoryStatements{i} = strcat([' ' 'clone Theories.NotEqualToZero
                        as' ' ' BlockNames{i} ' ' 'with function in1 =' ' ' get_param(in,
                        'Name') ', function out1 = ' get_param(out, 'Name') '\n\n']);
                end
            elseif isRequireBlock(BlockPaths{i})
                GoalStatements{GoalNumber} = strcat([' ' 'goal G' num2str(GoalNumber) ' '
                ': forall k: int.' ' ' get_param(in{1}, 'Name') ' ' 'k = True ->' ' '
                get_param(in{2}, 'Name') ' ' 'k = True ' '\n']);
                GoalNumber = GoalNumber + 1;
            end
        end
    end

    fprintf(fileID, '\n');
    for i = 1:NumberOfBlocks
        if ~isempty(TheoryStatements{i})
             fprintf(fileID, TheoryStatements{i});
        end
    end

    for i = 1:NumberOfRequireBlocks
        fprintf(fileID, GoalStatements{i});
    end

    fprintf(fileID,'end');
    fclose(fileID);
end


function is = isRequireBlock(Block)
    try
        %if the block has a require field, it is a requirement block
        param = get_param(Block, 'Require');
        is = true;
    catch
        is = false;
    end
end
```

```
function is = isCompareToZeroBlock(Block)
    try
        %if the block has a relop field, it is a compare to zero block
        rel = get_param(Block, 'relop');
        is = true;
    catch
        is = false;
    end
end

function is = isLogicBlock(Block)
    BlockType = get_param(Block, 'BlockType');
    if strcmp(BlockType, 'Logic')
        is = true;
    else
        is = false;
    end
end

function out = CapFirstLetter(str)
    out = strcat([upper(str(1)) str(2:length(str))]);
end

function num = FixToReal(str)
    val = str2double(str);
    if val < 0
        %add '-.' at the frot of the number if it is negative
        str = strcat(['-.' num2str(-val)]);
    end

    %calculate fractional part
    frac = abs(val) - floor(abs(val));

    if val == 0
        str = '0.0';
    elseif frac == 0
        %if there is no fractional part, add '.0' to make the value real
        str = strcat([str '.0']);
    end

    num = str;
end
```

Appendix B

B.1 Example of an M-circle exclusion region verification condition

In this appendix, we provide a verification criterion for the M-circle exclusion region (with an M value of $M = 10^{0.05}$) for the linear RECONFIGURE benchmark flight point 83 with a *Type 1 (T1)* backup controller in order to illustrate the complexity of the conjecture that the quantifier elimination algorithm is asked to prove.

```
∀ Y:(-1<=Y&Y<=1)=>
(4424773360497321226122986089102243231360963575314235289006084829170119408589697138
1244006941487060101549422880831841781095453527940086781*Y^2)/
17745086042373215101301850778515735701993197282405226081091069315933576369956003987
4558361990664932998233037501529828597054346100736
-
(1755209549564507361900158504292323824842586151549510439379599356737914012330295563
361159313242617054508353391399538260038277539240809054675*Y)/
45427420268475430659332737993000283397102585042957378767593137448789955507087370207
886940669610222847547657600391636120845912601788416
-
(2251712946153386503398276503900836096693569577977237581961396835556215840730340391
6555379465318618946610313243883644954931783829467488855*Y^3)/
22181357552966518876627313473144669627491496603006532601363836644916970462445004984
319795248833116624779129687691228574631793262592
+
(8098616188673189529132972035901043875844300311902967801024890400258673744243646708
658921441205102074334553859651878213066368589548661629*Y^4)/
27726696941208148595784141841430837034364370753758165751704795806146213078056256230
39974406104139578097391210961403571828974157824
-
(1739499783961449201982960908395138594627874405556539198887816865977225522270313069
1922057917677507443191630670713923086883209850631542179*Y^5)/
27726696941208148595784141841430837034364370753758165751704795806146213078056256230
39974406104139578097391210961403571828974157824
+
(7122051484871438748663538965260398182137379334706961488444838723849435577600042110
809331111794437960810610443736978593200537850788860211*Y^6)/
69316742353020371489460354603577092585910926884395414379261989515365532695140640575
9993601526034894524347802740350892957243539456
-
(8816651523938650424614145197429650654431631411897778958274890082557569625994207743
738289305608480423252208437371712600314275735197451425*Y^7)/
69316742353020371489460354603577092585910926884395414379261989515365532695140640575
9993601526034894524347802740350892957243539456
+
(7810102761967639855236774571233862865387500509336721233493622325533875409420118635
487293125560368422564790449948320656672107324648905159*Y^8)/
69316742353020371489460354603577092585910926884395414379261989515365532695140640575
9993601526034894524347802740350892957243539456
-
(7341159334814206409052341678656964124692393468215559665015231216647304394302171150
339321195823250332901591587148482712786893*Y^9)/
12608641982846233347929292832045956417625516566548942933743453889358630966879107395
65256520156317300505812095689818112
-
(2543059985672127513354456557308815510822566559571017306746119048465059400230250509
077373201455172555923923691152632731357*Y^10)/
49252507745493099015348800125179517256349674088081808334935366755307152214371513264
26783281860614455100828498788352
+
(2351005650258086618006259552444684715290769642268857660212074792863718477237036449
930882202524203830512195730854226583047*Y^11)/
6156563468186637376918600015647439657043709261010226041866920844413394026796439158O
3347910232576806887603562348544
-
(87274986680164298278679025771219755039976311136551113867687560192506241293385764O7
3518013254079294795076112525437606l*Y^12)/
3006134505950506531698535163890351395040873662602649434505332443561227552146698807б
33534717932503939880876769 28
+
(5557831311537703801699360732969306299376020610398306779977742331754127476797183592
01919360739723561032662150856026839*Y^13)/
7695704335233296721148250019559299571304636576262782552333651055516742533495548947 5
41848877907210086095044529356 8
+
(7670093169067681873569659540503268802928624157982989113389487105917952220465612319
43076870271405864736041263895329531 55*Y^14)/
3847852167616648360574125009779649785652318288131391276166825527758371266747774473 7
7092443895360504304752226467 84
-
(4159790383590646891512490762150412701538024129669221196457801893795679018136227136
407340059840099280362063984621664979 1*Y^15)/
1923926083808324180287062504889824892826159144065695638083412763879185633373887236 8
8546221947680252152376113233 92
+
(5167890582530298044627574818414094396794654501715643090866438292746624448173840611
250899077390232699793075813229638869 7*Y^16)/
3847852167616648360574125009779649785652318288131391276166825527758371266747774473 7
7092443895360504304752226467 84
-
(9544797070075152055314251271864083045929523770607432815179741482383782070599683030
7123088774498959216 17*Y^17)/
1708789628736728065916017364935641691682163617885322215957633286257775780624512440 0
183696695492608
+
(2525601418545376660318719496055821131138149288705342591333286627167667433428426697
242771262991*Y^18)/
15914343565113172548972231940698266883214596825515126958094847260581103904401068017
057792
-
(1884654105846703880286081749392033230248697292021264042382253422220934426408273250
036002352679*Y^19)/
63657374260452690195888927762793067532858387302060507832379389042324415617604272068
231168
+
(2560968632064730758607258480185059201603502681903814393519141318762950462066111101
2004063703*Y^20)/
79571717825565862744861159703491334416072984127575634790474236302905519522005340085
28896
-
(2188725274398604568382976171694723210769656965712174160295089423577636916212911979
053532097*Y^21)/
15914343565113172548972231940698266883214596825515126958094847260581103904401068017
057792
-
(22396003544819377276050818851931064060544386741321008002621450078656407047*Y^22)/
7067388259113537318333190002971674063309935587502475832486424805170479104
+
(7702956590613580054437489338860780494325142415*Y^23)/
12554203470773361527671578846415332832204710888928069025792
+
14164386114896600326945515557722709680597657457257847727298043111937282472133059203
3141980113572184673027434906520804695346970511262070925052599296025/
49947976805055875702105555676690660891977570282639538413746511354005947821116249921
924897649015871538557230897942505966327167610868612564900642816>0.
```

B.2 SAO implementation of the RECONFIGURE backup flight control law

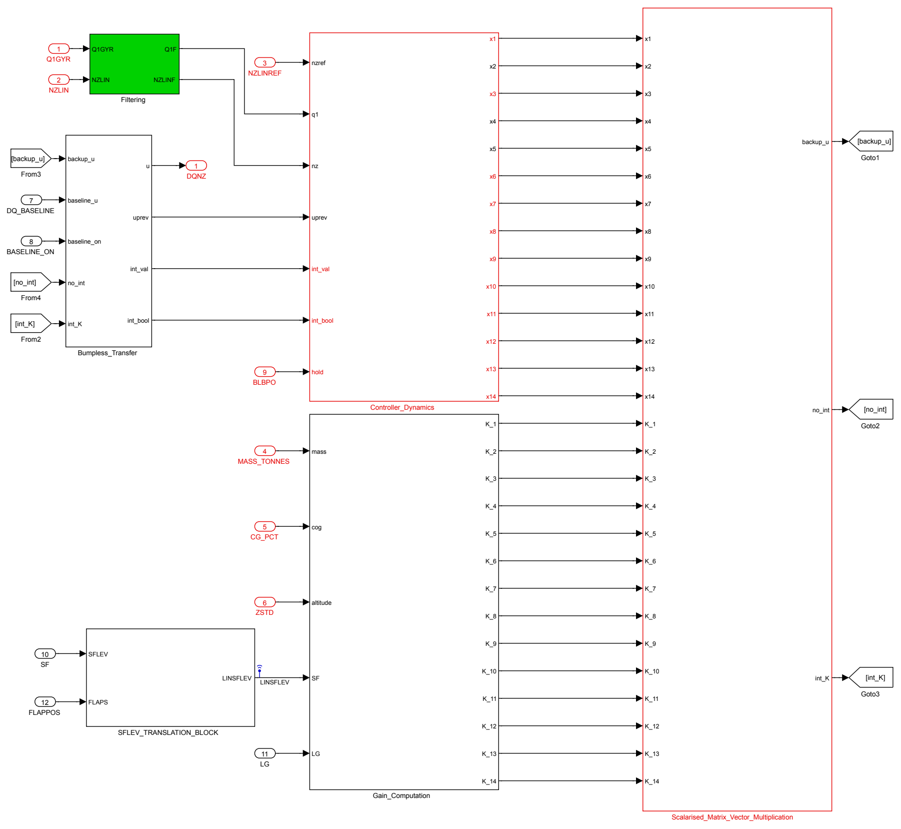


Fig. B.1 Vertical load factor n_z control law top level.

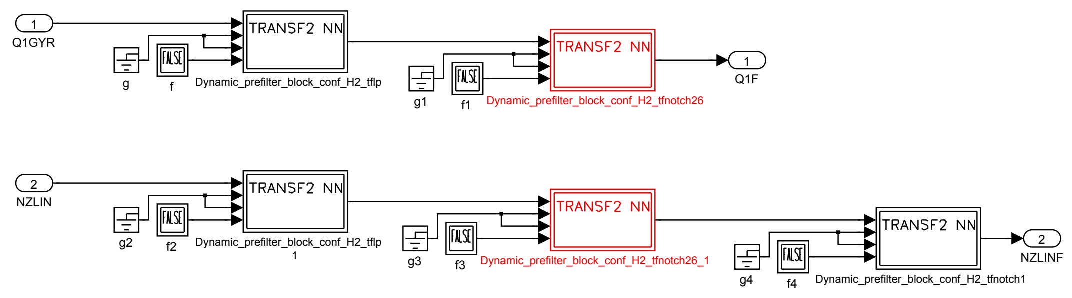

1
Q1GYR
TRANSF2 NN
g
f
Dynamic_prefilter_block_conf_H2_tflp
TRANSF2 NN
g1
f1
Dynamic_prefilter_block_conf_H2_tfnotch26
1
Q1F
2
NZLIN
TRANSF2 NN
g2
f2
Dynamic_prefilter_block_conf_H2_tflp 1
TRANSF2 NN
g3
f3
Dynamic_prefilter_block_conf_H2_tfnotch26_1
TRANSF2 NN
g4
f4
Dynamic_prefilter_block_conf_H2_tfnotch1
2
NZLINF

Fig. B.2 Filtering of pitch rate and vertical load factor signals.

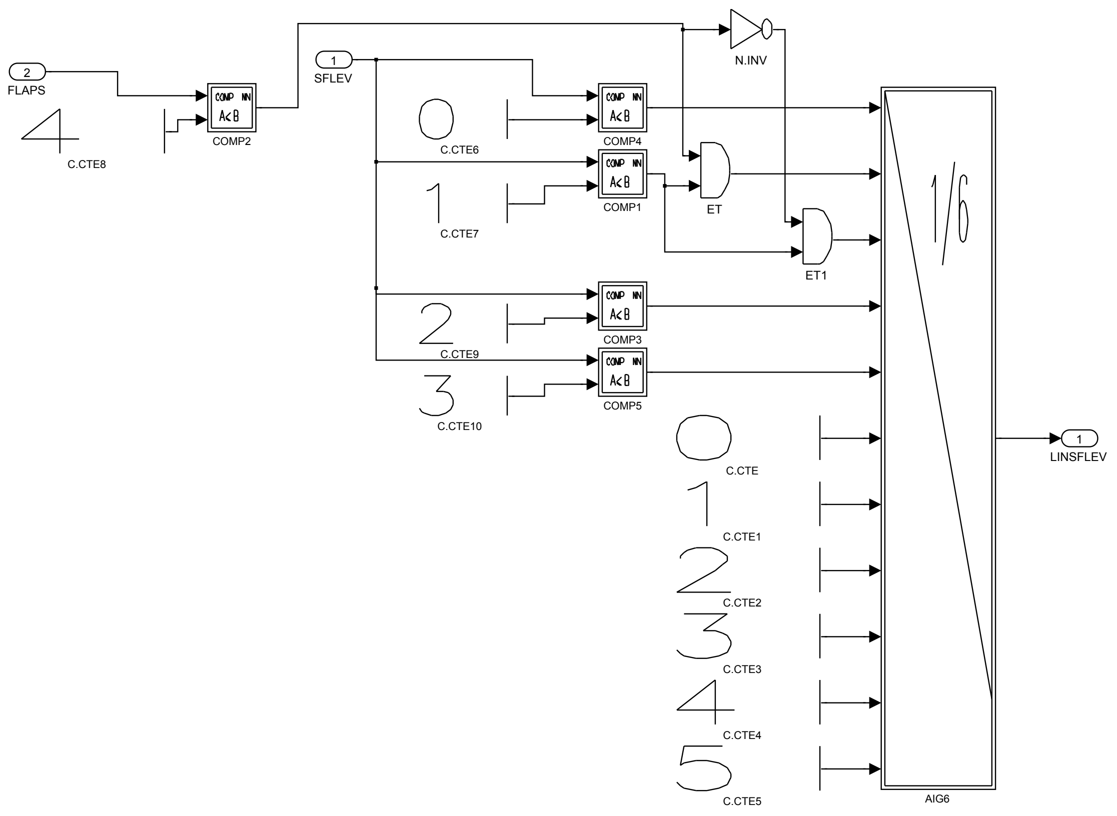


Fig. B.3 Vertical load factor n_z control law slat/flap lever setting translation block between nonlinear and linear simulator conventions.

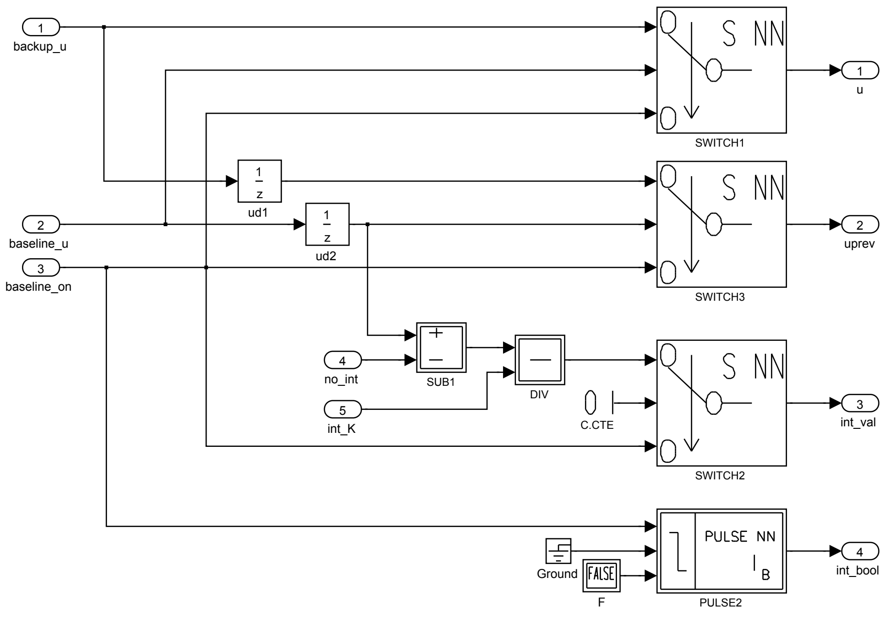


Fig. B.4 Bumpless transfer between Airbus baseline controller and the robust-to-lost-VCAS controller ensuring continuity of the elevator deflection input *u* at the moment of switching achieved by appropriately initialising integrator state *int_val*.

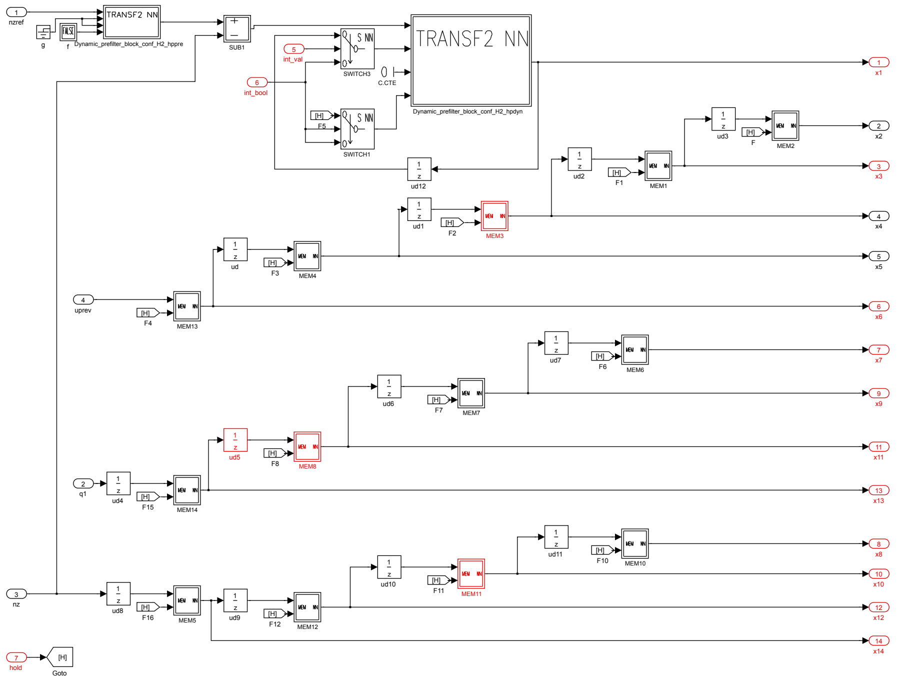


Fig. B.5 Vertical load factor n_z control law controller dynamics.

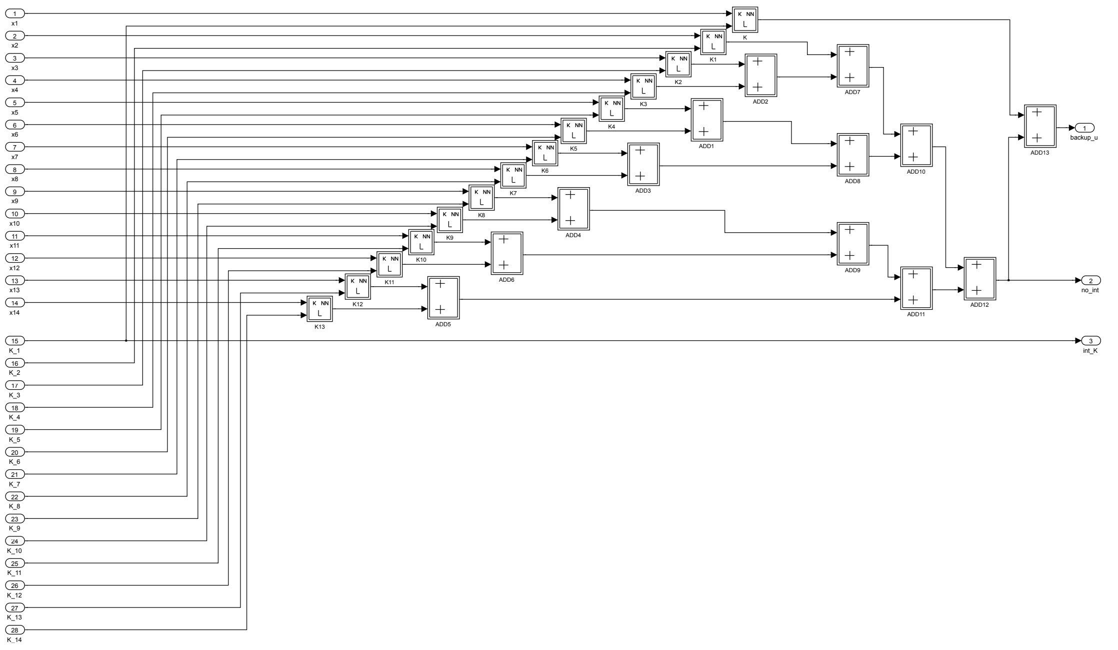


Fig. B.6 Vertical load factor n_z control law scalarised matrix-vector multiplication used to calculate elevator deflection input $u = Kx$.

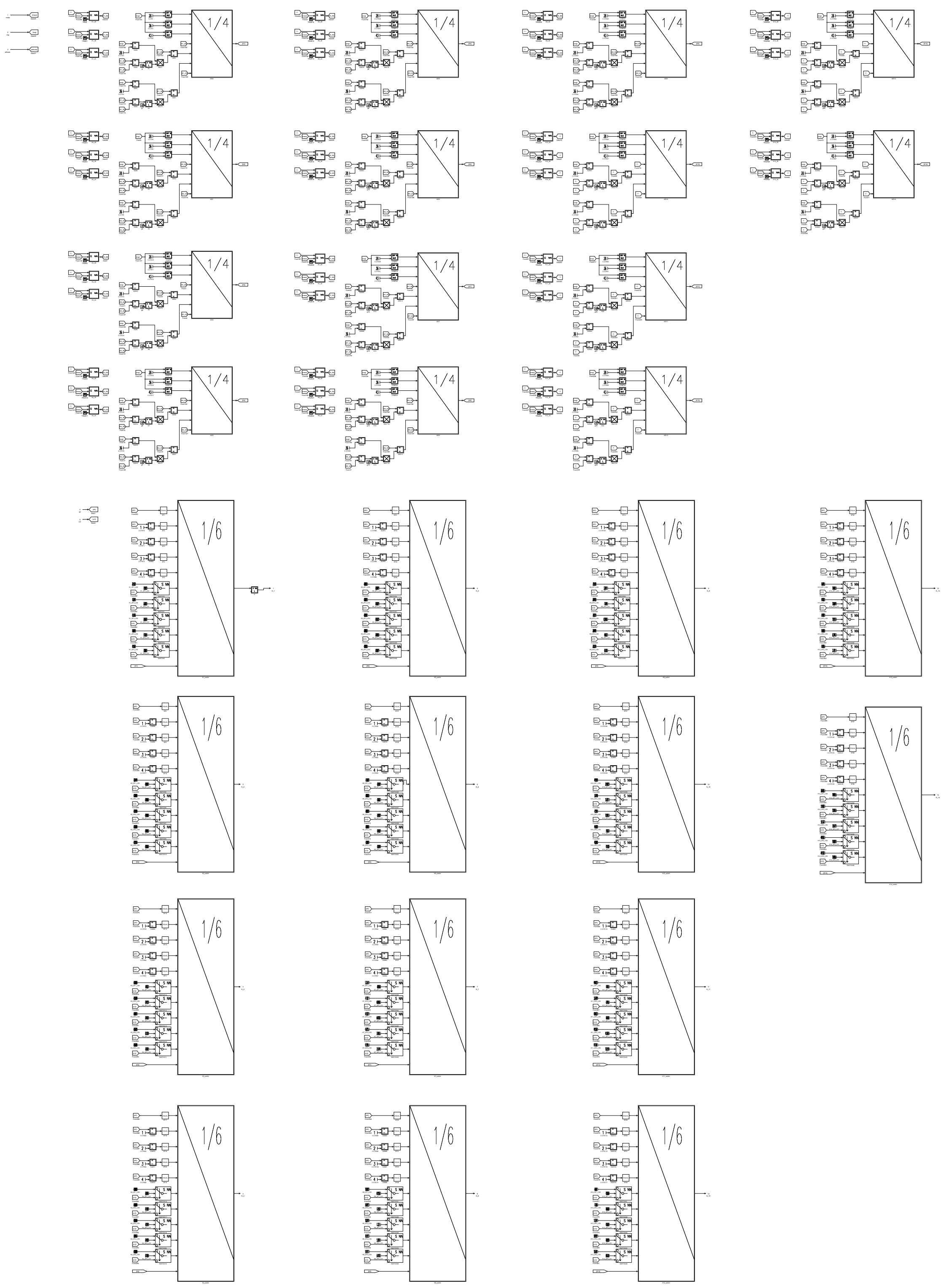


Fig. B.7 Vertical load factor n_z control law gain K computation via interpolation over mass, centre of gravity and altitude.