UNIVERSITÉ D'ANGERS

Année 2006
N° d'ordre: 798

QUANTIFIED REAL CONSTRAINT SOLVING USING MODAL INTERVALS WITH APPLICATIONS TO CONTROL

THÈSE DE DOCTORAT

Spécialité: Automatique et Informatique Appliqué

ÉCOLE DOCTORALE D'ANGERS

Présentée et soutenue publiquement

le : 22 Décembre 2006
à : Universitat de Girona (Girona – Espagne)
par : Pau HERRERO VIÑAS

Devant le jury ci-dessous :

Dr. Eric WALTER	**Président**	**Directeur de recherche au CNRS**
Pr. Frédéric BENHAMOU	**Rapporteur**	**Université de Nantes**
Dr. Stefan RATSCHAN	**Rapporteur**	**Academy of Sciences of the Czech Republic**
Pr. Rafael M. GASCA	**Examinateur**	**Universidad de Sevilla**
Dr. Jorge BONDIA	**Examinateur**	**Universidad Politécnica de Valencia**
Pr. Luc JAULIN	**Examinateur**	**ENSIETA**
Pr. Miguel Á. SAINZ	**Examinateur**	**Universitat de Girona**

Directeur de thèse : Pr. Luc JAULIN

Co-encadrant : Pr. Josep VEHÍ

Nom et coordonnées du Laboratoire :

LISA: Laboratoire d'Ingénierie des Systèmes Automatisés
Université d'Angers 62, avenue Notre Dame du Lac - 49000 Angers
Tél: +33 (0)2.41.22.65.60 - Fax : +33 (0)2.41.22.65.61

ED 363

Acknowledgements

This work has been supported by the FPI Research Grant BES-2004-6337 subject to the Spanish CICYT Project DPI2003-07146-C02-02.

I would also like to thank certain people for their inestimable support during these short years since, without their help, I would not have been able to finish my work.

First of all, I want to mention my first supervisor, Josep Vehí. He is the person who most encouraged me while working at the Modal Interval and Control Engineering Laboratory (MICELab). I also want to thank him for all the resource material he provided for my work. My most special gratitude to Luc Jaulin from ENSIETA: École Nationale Supérieure d'Ingénieurs (France), my second but no less important supervisor. He has guided my steps through these years and I have learnt so much from our interminable discussions. I also wish to express my sincere appreciation to Miguel A. Sainz, without his support this thesis would not be possible. I am also thankful to Joaquim Armengol for his clever advices along these years.

I am also grateful to the rest of the people from MICELab at the Universitat de Girona for their technical and "less technical" support. In the same way, I express my gratitude to the people from LISA laboratory at the Université d'Angers and people from E3I2 laboratory at the ENSIETA: École Nationale Supérieure d'Ingénieurs, where part of this work was developed during different research stages.

And last but not least, many thanks to the researchers world-wide with whom I have been enjoying many fruitful discussions and collaborations. Special thank goes to the ones who have helped me reviewing this manuscript.

Finally, I would like to dedicate this achievement to my family and my friends. They have always believed in me, a priceless motivation which I truly appreciate. I hope that I have not disappointed them.

Résumé

Les contraintes réelles quantifiées (QRC) forment un formalisme mathématique utilisé pour modéliser un très grand nombre de problèmes physiques dans lesquels interviennent des systèmes d'équations non linéaires sur des variables réelles, certaines d'entre elles pouvant être quantifiées. Les QRCs apparaissent dans nombreux contextes comme, l'Automatique, le Génie Électrique, le Génie Mécanique, et la Biologie.

La résolution de QRCs est un domaine de recherche très actif pour lequel deux approches radicalement différentes sont proposées: l'élimination symbolique de quantificateurs et les méthodes approximatives. Cependant, la résolution de problèmes de grandes dimensions et la résolution du cas général, restent encore des problèmes ouverts.

Dans le but de contribuer à la résolution de QCRs, cette thèse propose une nouvelle méthodologie approximative basée sur l'Analyse par Intervalles Modaux (MIA), une théorie mathématique développée par des chercheurs de l'université de Barcelone et de l'université de Girone. Cette théorie permet de résoudre d'une façon élégante une grande classe de problèmes dans lesquels interviennent des quantificateurs logiques sur des variables réelles.

Parallèlement, ce travail a comme but de promouvoir l'utilisation de l'Analyse par Intervalles Modaux pour résoudre des problèmes complexes, comme sont les QRCs. La théorie de MIA est relativement confidentielle du fait de sa complexité théorique relative et du fait d'une formulation mathématique peu usuelle. Cette thèse essaie de lever cette barrière en présentant la théorie d'une façon plus intuitive à travers des exemples et des analogies provenant de la théorie classique de l'analyse par intervalles.

La méthodologie proposée a été implémentée informatiquement et validée à travers la résolution de nombreux problèmes de la littérature, et les résultats obtenus ont été comparés avec différentes techniques de l'état de l'art. Enfin, il a été montré que l'approche présentée apporte des améliorations en étendant la classe de QRCs qui peut être traité et en améliorant les temps de calcul pour quelques cas particuliers.

Tous les algorithmes présentés dans ce travail sont basés sur un algorithme développé dans le cadre de cette thèse et appelé f^* algorithme. Cet algorithme permet la réalisation de calculs par intervalles modaux de façon très simple, ce qui aide à l'utilisation de la théorie de MIA et facilite sa diffusion. Dans le même but, un site Internet a été créé afin de permettre l'utilisation de la plupart des algorithmes présentés dans la thèse.

Finalement, deux applications à l'Automatique sont présentées. La première application faite référence au problème de la détection de défauts dans des systèmes dynamiques, laquelle a été validée sur des systèmes réels. La deuxième application consiste en la réalisation d'un régulateur pour un bateau à voile. Cette dernière a été validée sur simulation.

Abstract

A Quantified Real Constraint (QRC) is a mathematical formalism that is used to model many physical problems involving systems of nonlinear equations linking real variables, some of them affected by logical quantifiers. QRCs appear in numerous contexts, such as Control Engineering, Electrical Engineering, Mechanical Engineering, and Biology.

QRC solving is an active research domain for which two radically different approaches are proposed: the symbolic quantifier elimination and the approximate methods. However, solving large problems within a reasonable computational time and solving the general case, still remain open problems.

With the aim of contributing to the research on QRC solving, this thesis proposes a new approximate methodology based on Modal Interval Analysis (MIA), a mathematical theory developed by researchers from the University of Barcelona and from the University of Girona. This methodology allows solving in an elegant way, problems involving logical quantifiers over real variables.

Simultaneously, this work aims to promote the use of MIA for solving complex problems, such as QRCs. The MIA theory is relatively confidential due to its theoretical complexity and due to its non-conventional mathematical notation. This thesis tries to raise this barrier by presenting the theory in a more intuitive way through examples and analogies from the classical Interval Analysis approach.

The proposed methodology has been implemented and validated by resolving several problems from the literature, and comparing the obtained results with different state-of-the-art techniques. Thus, it has been shown that the presented approach extends the class of QCRs that can be solved and improves the computation time in some particular cases.

All the presented algorithms in this work are based on an algorithm developed in this thesis and called Fstar algorithm. This algorithm allows the computation with Modal Intervals in an easy way, something that helps to the utilization of MIA and facilitates its diffusion. With this purpose, an Internet site has been created to allow the utilization of most of the algorithms presented in this thesis.

Finally, two control engineering applications are presented. The first application refers to the problem of fault detection in dynamic systems and has been validated from experiments involving actual processes. The second application consists of the realization of a controller for a sailboat. This last one has been validated using simulation.

Resum

Les restriccions reals quantificades (QRC) formen un formalisme matemàtic utilitzat per modelar un gran nombre de problemes físics dins els quals intervenen sistemes d'equacions no-lineals sobre variables reals, algunes de les quals podent ésser quantificades. Els QRCs apareixen en nombrosos contextos, com l'Enginyeria de Control, l'Enginyeria Elèctrica, l'Enginyeria Mecànica, i la Biologia.

La resolució de QRCs és un domini de recerca molt actiu dins el qual es proposen dos enfocaments radicalment diferents: l'eliminació simbòlica de quantificadors i els mètodes aproximatius. Tot i això, la resolució de problemes de grans dimensions i la resolució del cas general, resten encara problemes oberts.

Amb la finalitat de contribuir a la resolució de QRCs, aquesta tesi proposa una nova metodologia aproximativa basada en l'Anàlisi Intervalar Modal (MIA), una teoria matemàtica desenvolupada per investigadors de la Universitat de Barcelona i de la Universitat de Girona. Aquesta teoria permet resoldre de manera elegant problemes en els quals intervenen quantificadors lògics sobre variables reals.

Simultàniament, aquest treball pretén promoure la utilització de la teoria de MIA per resoldre problemes complexes, com són els QRCs. La teoria MIA és relativament confidencial degut a la seva complexitat teòrica i a una notació matemàtica poc usual. Aquesta tesi pretén elevar aquesta barrera presentant la teoria d'una forma més intuïtiva mitjançant exemples i analogies provenint de la teoria clàssica de l'Anàlisi Intervalar.

La metodologia proposta ha estat implementada informàticament i validada mitjançant la resolució de nombrosos problemes de la literatura, i els resultats obtinguts han estat comparats amb diferents tècniques de l'estat de l'art. D'aquesta manera, s'ha mostrat que l'enfocament presentat aporta una sèrie de millores estenent la classe de QRCs que poden ser tractats i millorant el temps de càlcul en alguns casos particulars.

Tots el algoritmes presentats en aquest treball són basats en un algoritme desenvolupat en el marc d'aquesta tesi i que és anomenat algoritme Fstar. Aquest algoritme permet realitzar càlculs amb Intervals Modals de forma molt simple, la qual cosa ajuda enormement a la utilització de la teoria de MIA i facilita la seva difusió. Amb el mateix objectiu, s'ha creat una pàgina d'Internet que permet la utilització de la major part dels algoritmes presentats dins la tesi.

Finalment, dues aplicacions a l'Enginyeria de Control són presentades. La primera aplicació fa referència al problema de detecció de fallades en sistemes dinàmics i ha estat validada mitjançant experiments en processos reals. La segona aplicació consisteix en la realització d'un controlador per a un vaixell a vela. Aquesta última ha estat validada mitjançant simulació.

NOTATION

- ϕ: Quantified real constraint.
- Σ: Solution set of a quantified real constraint ϕ.
- x: Real value.
- X: Modal interval.
- X': Real domain or classic interval.
- $[\underline{a}, \overline{a}]$: Modal Interval, where $\underline{a}$ is the lower bound and $\overline{a}$ is the upper bound.
- $[\underline{a}, \overline{a}]'$: Real domain or classic interval.
- $\boldsymbol{x}$: Vector of real values.
- $\boldsymbol{X}$: Vector of modal intervals (a box).
- $\boldsymbol{X}'$: Vector of real domains or classic intervals.
- $\mathbf{A}$: Matrix of real values.
- $I\mathbb{R}$: Set of classic intervals.
- $I^*\mathbb{R}$: Set of modal intervals.
- f: Continuous function.
- $\mathbf{f}$: Vector of continuous functions.
- f^*: *-semantic extension of a continuous function f.
- fR: Rational modal interval extension of a continuous function f.
- $InnR(fR)$: Inner rounding affecting to the interval operations for computing fR.
- $OutR(fR)$ Outer rounding affecting to the interval operations for computing fR.

- MIA: Modal Interval Analysis.
- QRC: Quantified real constraint.

Modal Interval Analysis notation

The notation used in this thesis concerning Modal Interval Analysis (MIA) is slightly different from the original one. This change is motivated by the belief that the original notation, despite of having a mathematical justification, is both difficult to understand for non-expert readers and not strictly necessary from a practical point of view. Therefore, a more standard notation is used. The introduced change refers to the way of representing how a logic quantifier $\{\forall, \exists\}$ affects a variable which ranges over a real domain. MIA represents this assertion by

$$Q(x, X'),$$

where Q is the modal quantifier $Q \in \{U, E\}$, x is the real variable and X' the real domain. A justification of the use of this notation can be found in Gardeñes *et al.* (2001). However, for an easier comprehension of the document, this thesis uses a more standard notation. Therefore, the same statement is represented by

$$(Qx \in X'),$$

where Q is the logic quantifier $Q \in \{\forall, \exists\}$.

Contents

Nomenclature **xix**

1 Introduction **1**

1.1 Motivation 1

1.1.1 The importance of quantified real constraints 1

1.1.1.1 Example 2

1.1.2 The potentiality of Modal Interval Analysis 3

1.2 Objectives 5

1.3 Thesis organization 6

2 Quantified Real Constraint Solving **8**

2.1 Introduction 8

2.2 Problem definition 9

2.2.1 Solving a quantified real constraint 9

2.2.2 Specific problem definition 10

2.3 State-of-the-art 12

2.3.1 Quantifier elimination methods 12

2.3.1.1 A short history of quantifier elimination methods 13

2.3.1.2 An overview of QE by CAD 14

2.3.2 Approximate methods 16

2.3.2.1 Consistency of bounded quantified real constraint 17

2.3.2.2 Approximate solution set 21

2.3.2.3 Cylindrical Box Decomposition 26

2.3.2.4 Multivariate Bernstein Polynomials 27

2.3.2.5 Interval methods 27

2.3.2.6 Contractor methods 29
2.4 Conclusions 31

3 Quantified Real Constraint Satisfaction Using Modal Intervals 32
3.1 Introduction 32
3.2 Modal Interval Analysis 35
3.2.1 Basic concepts 35
3.2.1.1 Modal interval inclusion 36
3.2.1.2 Modal interval lattice operators 37
3.2.1.3 Modal interval arithmetic 38
3.2.1.4 *-Semantic extension 38
3.2.1.5 *-Semantic theorem 40
3.2.1.6 *-Semantic extension computation 41
3.2.1.7 Interpretable modal interval rational extension . 43
3.2.1.8 Modal interval inclusion test 47
3.3 f^* algorithm 50
3.3.1 Key theorem 51
3.3.2 Basic algorithm 56
3.3.2.1 Bounding criteria 59
3.3.2.2 Stopping criteria 59
3.3.3 Improvements 60
3.3.3.1 Selection strategy 60
3.3.3.2 Monotonicity study 61
3.3.3.3 Tree-optimality study 62
3.3.4 Step-by-step example 64
3.3.5 Complexity, termination, soundness and completeness . . 66
3.4 Examples 67
3.5 Conclusions 69

4 Quantified Set Inversion Using Modal Intervals 71
4.1 Introduction 71
4.1.1 Set Inversion Via Interval Analysis 71
4.2 Quantified Set Inversion algorithm 72
4.3 Application examples 78

4.3.1 Robust Control 78
4.3.2 Set projection 83
4.3.2.1 Application to computer graphics 84
4.3.3 Bounded-error Parameter Identification 85
4.3.3.1 Comparisons 88
4.3.4 Aircraft control 90
4.4 Conclusions 93

5 Continuous Minimax Optimization Using Modal Intervals 95
5.1 Introduction 95
5.2 Minimax optimization using modal intervals 98
5.2.1 The monotonic case 99
5.2.2 The non-monotonic case 101
5.3 Algorithm 103
5.3.1 Unconstrained version 103
5.3.1.1 Examples 105
5.3.2 Constrained version 107
5.3.2.1 Examples 111
5.4 Conclusions 113

6 Application to Fault Detection 114
6.1 Introduction 114
6.2 Analytical redundancy 115
6.2.1 Consistency test 117
6.2.2 Window consistency 118
6.2.3 Fault detection algorithm 120
6.2.4 Graphical output 121
6.3 Applications 122
6.3.1 PROCEL pilot plant 123
6.3.1.1 Testing scenarios 123
6.3.1.2 Mass balance model 125
6.3.1.3 Energy balance model 125
6.3.1.4 Testing results 126
6.3.2 Steam Generator pilot plant 128

6.3.2.1 Process description 128
6.3.2.2 Testing scenario 129
6.3.2.3 Process model 129
6.3.2.4 Testing results 130
6.3.3 Fluid Catalytic Cracking plant 132
6.3.3.1 Process description 132
6.3.3.2 Test scenario 132
6.3.3.3 Process model 133
6.3.3.4 Test results 134
6.4 Conclusions 134

7 Application to Sailboat Control **136**
7.1 Introduction 136
7.2 Control strategy 137
7.3 Sailboat control application 139
7.4 Sailboat model 140
7.5 Polar diagram of a sailboat 141
7.5.1 Transformation of the problem 143
7.5.2 Resolution 144
7.6 Feedback linearization control 144
7.7 Pre-compensator module 148
7.8 Simulation results 151
7.9 Conclusions 151

8 Implementation **153**
8.1 Implementation philosophy 153
8.2 Programming tools 154
8.2.1 C++ language 154
8.2.1.1 Code::Blocks and GCC compiler 154
8.2.2 Standard Template Library 155
8.2.3 Spirit Parser framework 155
8.2.3.1 Grammar definition 156
8.2.3.2 Matching procedure 157
8.2.3.3 Semantic actions 157

8.2.4 wxWidgets framework 158
8.3 FSTAR Solver 158
8.3.1 Numeric implementation 159
8.3.1.1 Modal interval arithmetic library 159
8.3.1.2 The branch-and-bound algorithm 161
8.3.2 Symbolic implementation 162
8.3.2.1 The parser 163
8.3.2.2 Symbolic differentiation 165
8.3.2.3 Tree-optimality study 166
8.4 QRCS Solver 167
8.5 MINIMAX Solver 169
8.6 QSI Solver 170
8.7 SQUALTRACK Solver 172
8.7.1 The parser 175
8.7.2 Analytical Redundancy Relations construction 176
8.7.3 Communication Process Interface 177
8.7.3.1 Offline CPI 178
8.7.3.2 Online CPI 180
8.7.4 Numerical and graphical outputs 180
8.8 Generic user interface 183
8.9 Modal Interval Remote Solver 183
8.10 Conclusions 186

9 Conclusions and Future Work 188

9.1 Conclusions 188
9.1.1 Contributions 189
9.2 Future work 191
9.2.1 Combining approaches 191
9.2.2 Modal interval constraint propagation 191
9.2.3 Solving the vectorial case 192
9.2.4 New applications 192
9.3 Related publications 192
9.3.1 Publications in journals 192

9.3.2 Publications in conferences 193
9.3.3 Publications in workshops 195

A Problem Definitions **196**
A.1 FSTAR Solver problems 196
A.2 QRCS Solver problems 197
A.3 QSI Solver problems 198
A.4 MINIMAX Solver problems 204
A.5 SQUALTRACK Solver problems 208

References **223**

List of Figures

1.1 The orientation of an aircraft with respect to the airflow. 3

2.1 Solution set of Example 2.2.1. 14
2.2 Box approximations for Σ of Example 2.2.1. 21
2.3 Branch-and-bound algorithm. 23
2.4 Paving for Σ of Example 2.2.1. 23
2.5 Branch-and-prune algorithm. 25
2.6 RSOLVER screenshot corresponding to Example 2.2.1. 30
2.7 Proj2D screenshot corresponding to Example 2.2.1. 31

3.1 *Strip* and *Cell* partitions. 52
3.2 f^* algorithm step-by-step example. 66

4.1 Graphical output for Example 4.2.1. 76
4.2 Graphical output for Example 4.2.2. 77
4.3 Approximation of Σ for robust closed-loop stability. 79
4.4 RSOLVER and Proj2D comparisons for robust closed-loop stability. 80
4.5 Uncertain linear time-invariant model 81
4.6 Approximation of Σ_c obtained with QSI algorithm. 81
4.7 Comparisons with Projection Algorithm, RSOLVER, AQCS solver. 82
4.8 Projection of Σ on the $\boldsymbol{x}$-plane. 83
4.9 Projection of a paraboloid on the x_1x_2-plane. 84
4.10 Projection of a) crosscap, b) hiperbolic paraboloid and c) quartic surfaces. 85
4.11 QSI output for the bounded-error parameter identification problem. 88
4.12 Extended-SIVIA, Proj2D and RSOLVER comparisons. 89

4.13 QSI output for the aircraft equilibrium problem. 93

5.1 Feasibility region and partitions. 109

6.1 SQUALTRACK solver graphical output. 122
6.2 Flowsheet of PROCEL. 124
6.3 Additional input flow to reactor 1. 127
6.4 Reactor 1 leakage. 127
6.5 Reactor 1 heater shutdown. 128
6.6 Flowsheet of the Steam Generator. 129
6.7 Boiler Leakage. 131
6.8 Schematic of FCC process. 132
6.9 Faulty response of the air flow to a setpoint change. 134

7.1 Control diagram. 138
7.2 Sailboat control scheme. 139
7.3 Sailboat model. 142
7.4 Polar diagram obtained with the QSI solver. 144
7.5 Existence algorithm. 151
7.6 Manoeuver to moor the sailboat inside a harbor. 152

8.1 FSTAR solver architecture. 160
8.2 Binary tree representation. 163
8.3 Binary tree representation for example from Table 8.8. 165
8.4 Binary tree after its evaluation. 166
8.5 Derivatives binary trees. 166
8.6 Tree-optimality study representation. 167
8.7 QSI graphical output. 173
8.8 SQUALTRACK solver icon. 174
8.9 SQUALTRACK solver architecture. 176
8.10 First order model representation. 178
8.11 First order model representation for a window length of 5. 179
8.12 SQUALTRACK solver online version. 181
8.13 SQUALTRACK solver graphical output. 182
8.14 Modal Interval Solver graphical user interface. 184

8.15 Modal Interval Remote Solver main page. 185
8.16 MIRS interface corresponding to the QSI solver. 186
8.17 MIRS graphical output for the QSI solver. 187

Chapter 1

Introduction

This chapter describes the motivation leading to the presentation of this thesis. The desired objectives and the methodology used are explained in brief. The chapter ends with a description of the structure and contents of this thesis.

1.1 Motivation

This thesis is devoted to the resolution of Quantified Real Constraints (QRCs) using Modal Interval Analysis (MIA). The realization of this thesis has been mainly motivated by the importance of QRCs in many engineering domains and the belief that MIA is a highly suitable tool for solving this type of problem.

1.1.1 The importance of quantified real constraints

A QRC is a mathematical formalism that allows for the representation of a wide class of numerical problems involving systems of nonlinear equations (or inequations) linking variables involving real numbers, some of which can be affected by logical quantifiers (∀, ∃). This formalism can be applied to model many real-life problems whose variables are affected by physical uncertainties (e.g. noise and perturbations) or that are subject to a regulation (e.g. control inputs). For example, QRCs appear in numerous contexts, such as Control Engineering Abdallah *et al.* (1996); Dorato (2000); Jaulin & Walter (1996); Jirstrand (1997), Electrical

Engineering Sturm (2000), Mechanical Engineering Ioakimidis (1999) and Biology Chauvin *et al.* (1994).

In general, the problem of solving QRCs is undecidable Tarski (1951), and is very difficult in special cases Davenport & Heintz (1988). Nevertheless, different approaches have been proposed to tackle this problem. The first approach, coming from mathematical foundations, is referred to as Quantifier Elimination Tarski (1951), and the second approach, coming from engineering research, is referred to as Approximate Methods and is mainly based on validated numerical techniques such as Interval Analysis Moore (1966) and Constraint Propagation Benhamou & Older (1997); SamHaroud (1995). Both approaches have been implemented and applied in different domains for solving small or mid-sized problems. However, they are still far away from solving large real applications in a reasonable computational time, due to the high complexity of the proposed algorithms. Other limitations concerning the current QRC solving techniques refer to aspects such as the class of QRCs that can be dealt with. For example, most of the existing approaches are limited to inequality predicates, or are restricted to only one type of quantification. Therefore, QRC solving is still an active research domain with several open problems to be solved.

1.1.1.1 Example

An important question in advanced aircraft applications is to know what the orientation (α, β) of an aircraft is with respect to the airflow, and this can be controlled by the admissible control-surface configurations (u_1, u_2, u_3) Jirstrand (1997). See Figure 1.1.

The aerodynamic moments acting over the aircraft, T_L, T_M, and T_N, are nonlinear functions of (α, β), which are the angles of attack and sideslip, respectively, the control-surface deflections, (u_1, u_2, u_3), which are the aileron, elevator, and rudder deflections, respectively, and a set of uncertain coefficients $(q_1, \dots, q_n)$ modeling the geometry and aerodynamics of the aircraft. The problem of finding if, for a given orientation (α, β) and for each value of the uncertain coefficient $(q_1, \dots, q_n)$, there exists a control-surface configuration, (u_1, u_2, u_3), for which the

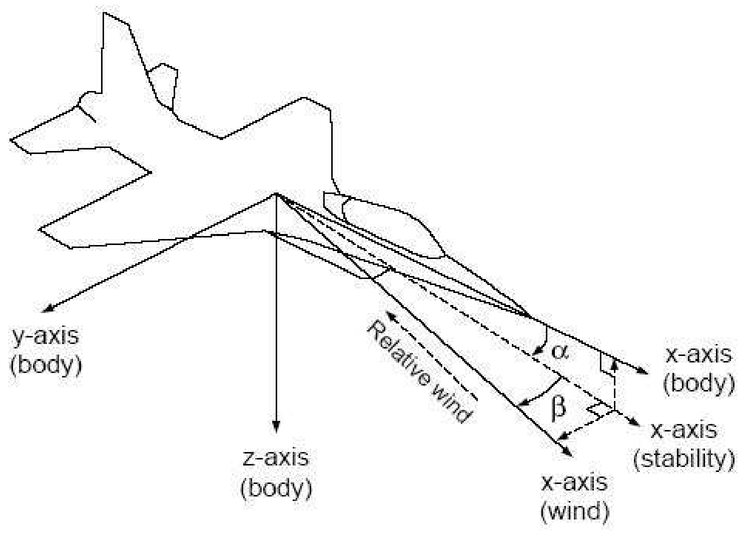


Figure 1.1: The orientation of an aircraft with respect to the airflow.

aerodynamic moments acting over the aircraft can be stabilized ($T_L = 0$, $T_M = 0$, $T_N = 0$), can be formulated as a QRC, which is mathematically represented as:

$$\begin{gathered}(\forall q_1 \in Q'_1) \dots (\forall q_n \in Q'_n)(\exists u_1 \in U'_1)(\exists u_2 \in U'_2)(\exists u_3 \in U'_3)\\ ((T_L(\alpha, \beta, \mathbf{q}, \boldsymbol{u}) = 0 \wedge T_M(\alpha, \beta, \mathbf{q}, \boldsymbol{u}) = 0 \wedge T_N(\alpha, \beta, \mathbf{q}, \boldsymbol{u}) = 0)\}, \end{gathered} \tag{1.1}$$

where $(Q'_1, \dots, Q'_n)$ and (U'_1, U'_2, U'_3) are the real domains in which the quantified variables are supposed to range. Thus, the set of orientation points, (α, β), for which the aircraft can be stabilized is:

$$\begin{gathered}\Sigma = \{(\alpha, \beta) | (\forall q_1 \in Q'_1) \dots (\forall q_n \in Q'_n)(\exists u_1 \in U'_1)(\exists u_2 \in U'_2)\\ (\exists u_3 \in U'_3)(T_L(\alpha, \beta, \mathbf{q}, \boldsymbol{u}) = 0 \wedge T_M(\alpha, \beta, \mathbf{q}, \boldsymbol{u}) = 0\\ \wedge T_N(\alpha, \beta, \mathbf{q}, \boldsymbol{u}) = 0))\}, \end{gathered} \tag{1.2}$$

which is the solution set of the corresponding QRC.

1.1.2 The potentiality of Modal Interval Analysis

In the previous decade, Interval Analysis (IA) Moore (1966) has become a successful tool used to deal with numerical problems in a guaranteed way. For example, problems such as solving systems of equations Neumaier (1990) or global optimization problems Hansen (1992) have been successfully tackled. IA was originally conceived to control numerical errors arising from computer arithmetic by encompassing real numbers with floating points intervals. It was quickly seen that this paradigm could be applied to deal with the physical uncertainty coming from the real world. Applications in different fields, such as electronics, economics, robotics, and automation, have been widely implemented Jaulin *et al.* (2001). By 1980, the theory of Modal Interval Analysis (MIA), developed by the SIGLA/X Group Gardeñes *et al.* (2001), was proposed to fill some of the existing gaps within IA theory. Some of these gaps were related to the algebraic structure of the existing theory, but the main contribution of MIA was its inherent capability to deal with a more complex class of problems involving logical quantifiers respecting an ordering (AE-quantification, where the universal quantifier precedes the existential quantifier). For example, proving the satisfaction of the QRC involved in the previous motivation example is beyond the scope of the standard tools provided by classic IA theory. On the other hand, MIA theory is able to deal with the same problem in a natural way. Thus, the belief that MIA is a highly suitable tool for solving QRCs has motivated the realization of this thesis.

MIA is a mathematical tool that provides a strong theoretical background to deal with problems involving uncertainty and logical quantifiers. Different works involving the application of MIA have been proposed. More specifically, in his thesis, Vehí Vehí (1998) proposed a methodology for designing robust controllers. In Armengol (1999), Armengol proposed an original approach to deal with the problem of robust fault detection involving dynamic systems. Finally, Calm presented an MIA approach to tackle the problem of simulation and control Calm (2006). Despite these works, the inherent theoretical complexity of MIA combined with a nonstandard mathematical notation, means that its use is not very extensive. Another motivation for this thesis is to overcome this barrier and to present MIA in a more practical way to spread its utilization and to show its potential for solving complex problems such as QRCs.

1.2 Objectives

The main objective of this thesis was to improve the limitations of the current state-of-the art techniques using an intensive application of theorems and tools provided by MIA.

The specific objectives of this thesis are as follows.

- To deal with QRCs involving AE-quantification (universal quantifier preceding the existential quantifier) and equality predicates. This class of QRCs cannot be efficiently solved by most of the current state-of-the art techniques Benhamou & Goualard (2000); Garloff (1993); Jaulin & Walter (1993); Jaulin *et al.* (2002); Ratschan (2003b). On the other hand, MIA can naturally solve this type of problem.

- To implement an efficient and numerically guaranteed algorithm for computing using MIA. To date, any efficient algorithm that exists for such a goal and the development of this type of algorithm is essential for the utilization of MIA. This type of algorithm is the core for all the QRC solving algorithms based on MIA and its efficiency is crucial for the performance of these algorithms.

- To improve the performance of existing QRC solving state-of-the-art techniques. These improvements must be verified by carrying out comparisons with normalized benchmarks; the objective refers to computation time and to the accuracy of the results. Nevertheless, we do not intend to improve the computation time in all cases, but only those where MIA is particularly well suited.

- To solve engineering applications to show the viability of the proposed approach for dealing with real engineering problems. The purpose of this objective was to show that the proposed approach was not limited to academic problems.

- To present MIA in a more intuitive and practical way to facilitate its understanding. As mentioned above, MIA presents an inherent theoretical complexity combined with a nonstandard mathematical notation. Thus, effort must expended to facilitate its understanding.

- To implement the proposed MIA-based algorithms, and to present them in the form of solvers to allow their easy utilization. To spread the utilization of MIA and its application to QRC solving, it is important to facilitate its use by means of software applications that have user-friendly interfaces and do not require a deep knowledge of the theory. With the same objective, the aim was to create a Web page containing all the developed MIA-based algorithms to spread their utilization.

1.3 Thesis organization

This document is structured into an introduction and eight chapters, with an appendix and a bibliography section at the end.

- Chapter 1, is this Introduction section, and presents an outline of the work and provides a justification for the work developed in this thesis.
- Chapter 2 formalizes the problem of QRS solving and surveys the different existing state-of-the-art techniques used to solve it.
- Chapter 3 describes a new technique based on MIA for testing the consistency of a class of QRC. This chapter provides a detailed description of an original algorithm for computing modal interval extensions of continuous functions. This algorithm is referred to as f^* algorithm and is one of the main contributions of this thesis.
- Chapter 4 presents an algorithm for finding the inner and outer approximations of the solution set of a class of QRC and solves several examples, comparing the results with some of the state-of-the-art techniques. This algorithm is referred to as the Quantified Set Inversion (QSI) algorithm and is another important contribution of this thesis.

- In Chapter 5, an original algorithm for solving continuous minimax optimization problems based on MIA is presented. This work discussed in this chapter is not related to QRC solving and can be seen as a collateral result of the f^* algorithm presented in Chapter 3.
- Chapter 6 presents an application of the proposed approach to the problem of detecting faults in dynamic systems. Experimental tests involving actual process data were carried out to validate the proposed approach.
- Chapter 7 explains an original application to the control of a sailboat. Simulation results were used to provide validation of the proposed approach.
- In Chapter 8, details on the implementation of the algorithms presented in this thesis are provided. A Web page that allows for the utilization of these algorithms through Internet access is also presented.
- Finally, Chapter 9 concludes this thesis by stating the main contributions of this work and suggesting further work. A list of related publications is included at the end of this chapter.
- In the Appendix A, the sources for introducing some of the examples provided in the thesis to the corresponding software implementation are provided.

Chapter 2

Quantified Real Constraint Solving

This chapter provides a description of the problem we want to solve, as well as a review of the existing techniques to deal with this type of problems.

2.1 Introduction

A quantified constraint is a first-order formula Ebbinghaus *et al.* (1984) that contains logical quantifiers ($\exists, \forall$), logical connectives ($\wedge$, $\vee$, $\neg$, $\Rightarrow$), predicate symbols (e.g.,$=$,$<$,$\leq$), function symbols (e.g. $+$, $-$, $\div$, $\times$), constants and variables ranging over discrete domains (e.g. boolean or integer domains) Bordeaux & Monfroy (2002); Gent & Walsh (1999) or over continuous domains (e.g. real domains) Benhamou & Goualard (2000); Collins (1975); Ratschan (2003c). Many problems arising in Artificial Intelligence or Engineering can be modeled as a quantified constraint. For instance, Game Theory and Scheduling are typical problems that can be stated as discrete quantified constraints, while Control Engineering and Robotics are fields where continuous quantified constraints can usually be applied. This thesis is focussed on the resolution of a particular class of continuous quantified constraints, denoted as Quantified Real Constraints (QRCs), where variables range over the real numbers.

2.2 Problem definition

In this thesis, we are interested in developing theories, algorithms and software for solving the following problem:

- **GIVEN:** A *quantified real constraint* ϕ,
- **FIND:** The *truth-value* of ϕ or the *solution set* of ϕ,

where a quantified real constraint (QRC) is a first-order formula Ebbinghaus *et al.* (1984) which variables range over reals numbers. From the set of involved variables, we can distinguish between quantified variables ($\boldsymbol{p}$), also referred to as parameters, and non-quantified variables ($\boldsymbol{x}$), also referred to as free-variables.

Example 2.2.1. *An example of QRC is*

$$(\forall p)\ p^2 + x_1 p + x_2 > 0, \tag{2.1}$$

where p is a quantified variable and x_1, x_2 are free-variables.

■

2.2.1 Solving a quantified real constraint

Solving a QRC can be understood in two different ways: Firstly, we can ask for its *truth-value* by asking the following question: Is a QRC true (or false) for whatever the values of the free-variables? And secondly, we can ask for the set of instantiations for the free-variables that make a QRC to be true. The second question corresponds to the notion of solution set of a QRC and is defined by

$$\Sigma = \{\boldsymbol{x} \in \mathbb{R}^n |\ (\mathbf{Q}\boldsymbol{p})\ \boldsymbol{c}(\boldsymbol{x}, \boldsymbol{p})\ \textit{is true}\}, \tag{2.2}$$

where $\mathbf{Q}$ is a vector of logical quantifiers ($\mathbf{Q}_i \in \{\forall, \exists\}$) and $\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{p})$ is the vector of real predicates, linked by means of logical connectives, and linking the vectors of free-variables ($\boldsymbol{x}$) and parameters under consideration ($\boldsymbol{p}$).

Example 2.2.2. *Consider the quantified real constraint from Example 2.2.1. It is easy to prove that it is not true for whatever the values of the free-variables. Nevertheless, it is possible to ask for its solution set, which is given by*

$$\Sigma = \{\boldsymbol{x} \in \mathbb{R}^2 | (\forall p)\ p^2 + x_1 p + x_2 > 0\}. \tag{2.3}$$

■

The definition of solution set given by Equation 2.2 is not usually useful from a practical point of view, because it is not easy to prove using standard mathematics, that a point $\boldsymbol{x}$ belongs to the solution set Σ. Then, QRC solving is aimed to obtain more useful representations of Σ. It is important to note that, efficiently solving the most general definition of a QRC, remains an open problem and, up to now, only particular instances have been solved. This thesis is not focused on the resolution of the general problem, but on a special instance of QRCs.

2.2.2 Specific problem definition

This thesis is concerned with the resolution of QRCs where:

- The universal logical quantifiers ($\forall$) precede the existential ($\exists$) quantifiers ($\forall\exists$-quantification).
- The involved real predicates are expressed by a finite combination of elementary operators and functions such as $+, -, *, \sin, \cos, \ldots$ and relational operators such as $=, >, \geq, \ldots$.
- In general, equality predicates must not share existentially quantified variables with other predicates. However, this limitation can be sometimes overcome using symbolic transformations of the equality predicates to eliminate shared existentially quantified variables.

- The ranges of the involved variables need to be bounded by real domains.

Thus, the corresponding solution set for this kind of QRCs is defined by

$$\Sigma_{\forall\exists} = \{\boldsymbol{x} \in \boldsymbol{X}' \mid (\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{V}')\ \boldsymbol{c}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v})\}, \tag{2.4}$$

where $(\boldsymbol{u}, \boldsymbol{v})$ corresponds to the split vector of parameters $\boldsymbol{p} = (\boldsymbol{u}, \boldsymbol{v})$ into their components respectively affected by the universal and existential quantifiers, $(\boldsymbol{X}', \boldsymbol{U}', \boldsymbol{V}')$ are the vectors of associated real domains, $\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v})$ is the vector of real predicates, linked by means of logical connectives, and linking the vectors of free-variables and parameters under consideration.

Remark 2.2.1. *The restrictions concerning the quantifiers ordering, the nature of the involved predicates and the bounding of the variables' domains, are usual restrictions of techniques based on* approximate methods *(see Section 2.3.2).*

■

Example 2.2.3. *An example of QRC intended to be solved in this thesis is*

$$(\forall u \in U')(\exists v \in V')\ [u^2 + x_1 u + x_2 v = 0 \wedge x_1 u + sin(x_2) > 0]. \tag{2.5}$$

An example of QRC beyond the scope of this thesis is

$$(\exists v \in V')(\forall u \in U') x + u + v > 0, \tag{2.6}$$

because of the order of quantifiers. However, its negation

$$\neg((\exists v \in V')(\forall u \in U') x + u + v > 0) \Leftrightarrow (\forall v \in V')(\exists u \in U') x + u + v \leq 0, \tag{2.7}$$

can be solved. Another example beyond the scope QRC is

$$(\exists v_1 \in V_1')(\exists v_2 \in V_2')[-x_1 + x_2 v_1 = 0 \wedge -x_2 + (1 + x_1^2) v_1 + v_2^3 = 0], \tag{2.8}$$

because the existentially quantified variable v_1 is shared by the two involved equality predicates linked by a logical conjunction. However, by isolating v_1 in the first *predicate and substituting it into the second predicate, the resulting equivalent QRC is*

$$(\exists v_2 \in V_2')[-x_2 + (1 + x_1^2)x_1/x_2 + v_2^3 = 0], \tag{2.9}$$

which can be tackled in this thesis.

■

Although the mentioned restrictions, a wide range of practical importance problems can still be represented by this constrained formulation.

2.3 State-of-the-art

Until today, research on the problem of QRC solving has mainly been done from two different approaches:

- The first approach, Quantifier Elimination (QE) methods, coming from research in the foundations of mathematics, provides a framework for solving QRCs in an exact way. However, this approach is restricted to addition and multiplication as function symbols and its double-exponential computational complexity limits it applicability to very simple problems.

- The second approach, Approximate Methods, coming from engineering research, instead of obtaining the exact solution, try to approximate the solution of a QRC by using techniques based on validated numerical methods Benhamou & Older (1997); Garloff (1985); Moore (1966). Although these algorithms can deal with more complex problems than the quantifier elimination approach, they are usually restricted to special cases (e.g. quantifiers only occur once, only one type of quantifiers, linear constraints, inequality constraints).

2.3.1 Quantifier elimination methods

The Quantifier Elimination (QE) problem for solving QRCs consists of:

- **GIVEN:** A quantified real constraint ϕ,
- **FIND:** A quantified real constraint $\overline{\phi}$ in which no variable is quantified such that $\phi \Leftrightarrow \overline{\phi}$.

Remark 2.3.1. *In QE theory, QRCs are referred to as first-order logic formulas over the reals.*

■

Example 2.3.1. *An illustration of the purpose of QE can be found in the following basic problem. Consider the QRC of Example 2.2.1*

$$\phi : (\forall p)\ p^2 + x_1 p + x_2 > 0, \tag{2.10}$$

It is easy to see that not all the values of x satisfy the above-mentioned formula. QE eliminates the quantified parameter $\boldsymbol{p}$ and provides the following quantifier-free equivalent formula

$$\overline{\phi} : x_1^2 - 4x_2 < 0. \tag{2.11}$$

Figure 2.1 shows a graphical representation of the corresponding solution set, where the red area corresponds to the solution set and the blue area to the complementary set.

■

2.3.1.1 A short history of quantifier elimination methods

The quantifier elimination method finds its origin in a paper from Tarski published in 1951: "A decision method for elementary algebra and geometry" Tarski (1951). Tarski's purpose was to prove that the elementary theory for reals was decidable and to give a method in order to decide whether a sentence of this

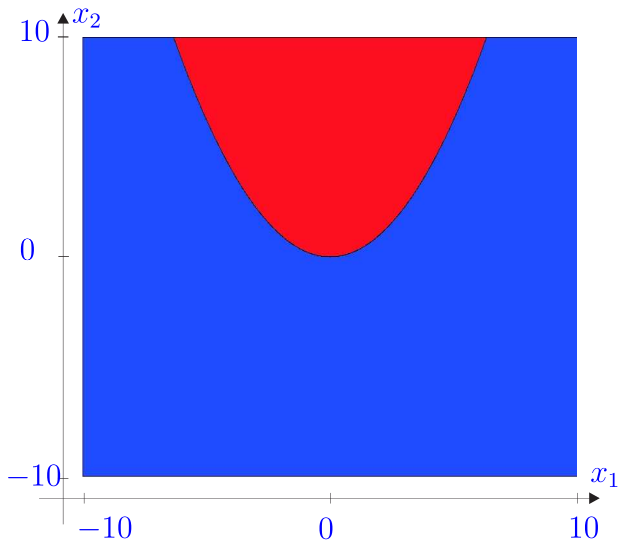


Figure 2.1: Solution set of Example 2.2.1.

theory was true or not. Tarski wanted to design a method such that one needs no "intelligence" to apply it, that could be applied step-by-step following the rules he gave. In his paper, Tarski presented a few examples of the application of his method. He found the solutions rather due to his mathematical intuition than by applying exactly his own method. The problem was that the method needed too many computation steps in order to be actually efficient because of its inherently doubly exponential complexity Davenport & Heintz (1988) in the number of variables. Therefore, no efficient implementation seems to be available. One of the first algorithms to solve the quantifier elimination-problem which is not only theoretically of a lower complexity but also allows more efficient implementations appeared only in 1975: in his paper "Quantifier elimination for real closed fields by cylindrical algebraic decomposition" Collins (1975), G. Collins presented an algorithm serving Tarski's purpose, which is based on a new method: the Cylindrical Algebraic Decomposition (CAD).

2.3.1.2 An overview of QE by CAD

The QE by CAD algorithm can be described as consisting of three phases:

1. **Projection.** One starts out with the set P_r of polynomials in r variables, which is extracted from the input ϕ . In $r-1$ projection steps there are $r-1$ further finite sets $P_{r-1}, \dots, P_1$ of polynomials with one fewer variable in each step generated.

2. **Extension.** Based on the trivial decomposition D_0 of 0-space one successively constructs decompositions $D_1, \dots, D_r$ of higher-dimensional spaces. For obtaining D_{i+1} there are sample points of D_i plugged into the polynomials of P_{i+1}. This yields univariate polynomials with real algebraic numbers as coefficients. Essentially, the roots of these polynomials and rational points between these roots extend the sample point of the underlying cell to give new sample points for cells in D_{i+1}.

3. **Solution formula construction.** Truth values are computed for the leaf cells of the tree, i.e. the cells in D_r. Depending on the quantifiers, these values are propagated down to cells in D_r, ..., D_k. Based on signs of projection polynomials, a quantifier-free solution formula is constructed to describe the subset of P_k which is comprised of true cells.

Several improvements of Collins' algorithms have been found since then, and considerable speed-ups could also be achieved by improvements in the implementations. One of the best implementations is the one by Hong, which is based on his thesis Hong (1990). Hong's quantifier-elimination-program is regularly updated by Brown and is available from Brown (2004). This implementation is quite usable for many small, as well as few mid-size problems. However it usually cannot finish big-size problems.

Further problems that occur when trying to solve QRCs exactly, include:

- They cannot efficiently deal with inputs containing uncertain parameters which are only known to be elements of an interval.

- Very often, the output consists of highly complicated algebraic expressions useless for many applications and thus requiring further processing in order to be useful.

- When they are interrupted before computing the total result, they do not provide a partial information. This limitation can be an important drawback on real time applications.

Despite of the mentioned limitations, QE methods have been applied to an important number of applications involving small and middle-size problems. For instance, there are applications in fields like Control Engineering Abdallah *et al.* (1999); Dorato *et al.* (1997); Jirstrand (1997) and Biology Chauvin & Müller (1951). In the last years, research on more efficient symbolic methods has mainly been concentrated on solving special cases Hong (1993); MacCallum (1988).

2.3.2 Approximate methods

Approximate methods try to avoid some of the deficiencies of QE methods by restricting oneself to approximate instead of finding the exact solution. These methods use validated numerical techniques, like Interval Analysis Moore (1966), Interval Constraint Propagation Benhamou & Older (1997); Davis (1975); Sam-Haroud (1995), Multivariate Bersnstein Polynomials Garloff (1985, 1993), or Interval Taylor Models Makino & Berz (2003) to provide approximate but guaranteed results. It should be noted that, in contrast to QE methods, Approximate Methods require *a priori* bounds on the range of the variables.

Thus, the input/output specifications of the problem to be solved by approximate methods are:

- **GIVEN:** A *bounded quantified real constraint* ϕ and $\epsilon \in \mathbb{R}^+$,
- **FIND:** The *truth-value* of ϕ or an *approximate solution set* of ϕ with error smaller than ϵ,

where a *bounded quantified real constraint* is a QRC which variables range on bounded real domains, an *approximate solution set* can be understood as an inner Σ_{Inn} or an outer Σ_{Out} approximation of the solution set of ϕ, such that,

$$\Sigma_{Inn} \subseteq \Sigma \subseteq \Sigma_{Out}, \tag{2.12}$$

and ϵ is an error bound, provided by the user, for which the algorithm solves the problem up to this error bound.

2.3.2.1 Consistency of bounded quantified real constraint

To obtain the consistency (truth-value) of a bounded QRC, two different approaches can be distinguished. The first approach groups these techniques which allows to approximate the range of a function over its variables' domain by means of an *inclusion function*. Once this approximation is obtained, an *inclusion test* can be performed to define the consistency of the QRC. This first approach includes methods based on validated numerical techniques like Multivariate Bersnstein Polynomials and Interval Analysis. The second approach, which includes validated numerical techniques like Interval Constraint Propagation, proves the consistency of a QRC by disproving its negation, using the so-called notion of *contractor* or *narrowing operator* Benhamou *et al.* (1999), applied to prune the search space from non solution parts.

The inclusion test approach:

Definition 2.3.1. *Consider a function f from $\mathbb{R}^n$ to $\mathbb{R}$. The function F from $I\mathbb{R}^n$ to $I\mathbb{R}$ is an inclusion function for f on a given box $\boldsymbol{X}'$ if and only if*

$$(\forall \boldsymbol{x} \in \boldsymbol{X}') f(\boldsymbol{x}) \in F(\boldsymbol{X}'), \tag{2.13}$$

where F can be obtained by means using different validated numerical techniques.

■

Example 2.3.2. *To illustrate the notion of inclusion function, consider the function f from $\mathbb{R}^3$ to $\mathbb{R}$,*

$$f(p, x_1, x_2) = p^2 + x_1 p + x_2, \tag{2.14}$$

with variables that vary within $(p, x_1, x_2) \in [-10, 10]'^3$. An inclusion function for f on $[-10, 10]'^3$ can be obtained using Interval Arithmetic Moore (1966) by simply *replacing the real variables by their associated intervals and the real operators by their interval counterparts. Thus,*

$$F([-10,10]',[-10,10]',[-10,10]') =$$
$$[-10,10]'^2 + [-10,10]' * [-10,10]' + [-10,10]' = [-110,210]', \quad (2.15)$$

is an outer approximation of the exact range of f on $[-10,10]'^3$, which is $[-35,210]'$. This overestimation of the result is due to the well known dependency problem *in Interval Analysis.*

■

Consider a QRC of the form

$$(\forall \boldsymbol{u} \in \boldsymbol{U}') f(\boldsymbol{x},\boldsymbol{u}) \ \geq 0, \quad (2.16)$$

where $\boldsymbol{u}$ is a vector of universally quantified variables ranging over a box $\boldsymbol{U}'$, $\boldsymbol{x}$ is a vector of free-variables ranging on a box $\boldsymbol{X}'$ and f is a continuous function. Notice that proving the consistency of the previous QRC is equivalent to looking for the truth-value of the next logical statement,

$$(\forall \boldsymbol{x} \in \boldsymbol{X}')(\forall \boldsymbol{u} \in \boldsymbol{U}') f(\boldsymbol{x},\boldsymbol{u}) \leq 0, \quad (2.17)$$

something that can be easily done using the next *inclusion test.*

Definition 2.3.2. *An* inclusion test *for Equation 2.17 is defined by,*

$$T(\boldsymbol{X}',\boldsymbol{U}') = \begin{cases} true & if\ F(\boldsymbol{X}',\boldsymbol{U}') \subseteq [0,\infty)', \\ false & if\ F(\boldsymbol{X}',\boldsymbol{U}') \cap [0,\infty)' = \emptyset, \\ undefined & otherwise, \end{cases} \quad (2.18)$$

where $F(\boldsymbol{X}',\boldsymbol{U}')$ is an inclusion function of $f(\boldsymbol{x},\boldsymbol{u})$ on $(\boldsymbol{X}',\boldsymbol{U}')$.

■

Example 2.3.3. *To illustrate the notion of inclusion test, let us consider the QRC from Example 2.2.1. For a given box* $(x_1, x_2) \in ([-0.5, 0.5]', [9, 10]')$ *and an interval bound for the involved quantified parameter* $p \in [-10, 10]'$*, the following logical statement has to be tested to prove the truth-value of the corresponding QRC,*

$$(\forall x_1 \in [-0.5, 0.5]')(\forall x_2 \in [9, 10]')(\forall \boldsymbol{p} \in [-10, 10]')\ p^2 + x_1 p + x_2 > 0. \qquad (2.19)$$

Then, an inclusion for the corresponding function computed using Interval Arithmetic is

$$\begin{aligned} &F([-0.5, 0.5]', [9, 10]', [-10, 10]') = \\ &[-10, 10]'^2 + [-0.5, 0.5]' * [-10, 10]' + [9, 10]' = [4, 115]'. \end{aligned} \qquad (2.20)$$

As the obtained result is included in $[0, \inf)'$*, the inclusion test return true and the QRC is satisfied.*

■

The contractor approach: A *contractor* $\mathcal{C}_\Sigma$, or *narrowing operator*, Benhamou *et al.* (1999) is any algorithm which allows to eliminate, in a guaranteed way, parts of the search space $\boldsymbol{X}'$ which do not belong to the solution set Σ of a QRC. Therefore, if the whole space of $\boldsymbol{X}'$ is eliminated, the solution set of QRC is proven to be empty. By negating QRC, it is possible to build a contractor for its complementary solution set $\neg\Sigma$, which eliminates parts belonging to the solution set. Consequently, if the whole space of $\boldsymbol{X}'$ is eliminated by the $\mathcal{C}_{\neg\Sigma}$, the QRC is proven to be true on $\boldsymbol{X}'$.

Formally speaking, a contractor can be defined by

Definition 2.3.3. *A* contractor *for the set* Σ*, is an operator* $\mathcal{C}_\Sigma : I\mathbb{R}^n \rightarrow I\mathbb{R}^n$

such that satisfies

$$\forall \mathbf{X}' \in I\mathbb{R}^n, \begin{cases} \mathcal{C}_\Sigma(\mathbf{X}') \subset \mathbf{X}' & \textit{(contractance)}, \\ \mathcal{C}_\Sigma(\mathbf{X}') \cap \Sigma = \mathbf{X}' \cap \Sigma & \textit{(completeness)}. \end{cases} \tag{2.21}$$

$\mathcal{C}_\Sigma$ *is* idempotent *if for all* $\mathbf{X}'$, $\mathcal{C}_\Sigma(\mathcal{C}_\Sigma(\mathbf{X}')) = \mathcal{C}_\Sigma(\mathbf{X}')$. *It is* thin *if for any singleton* $\{\mathbf{x}\}$, $\mathcal{C}_\Sigma(\{\mathbf{x}\}) = \{\mathbf{x}\} \cap \Sigma$. $\mathcal{C}_\Sigma$ *is said to be* convergent *if for almost any point* $\mathbf{x}$, *and for all sequences of nested boxes* $\mathbf{X}(k)$,

$$\mathbf{X}'(k) \to \mathbf{x} \Rightarrow \mathcal{C}_\Sigma(\mathbf{X}'(k)) \to \{\mathbf{x}\} \cap \Sigma. \tag{2.22}$$

It is said to be minimal *if*

$$\forall \mathbf{X}' \in I\mathbb{R}^n, \mathcal{C}_\Sigma(\mathbf{X}') = [\mathbf{X}' \cap \Sigma], \tag{2.23}$$

where $[\mathbf{X}' \cap \Sigma]$ *denotes the smallest box containing* $\mathbf{X}' \cap \Sigma$.

■

Thus, a consistency test for the QRC from Equation 2.16, is defined by

$$T(\boldsymbol{X}') = \begin{cases} true & if\ \mathcal{C}_{\neg\Sigma}(\boldsymbol{X}') = \emptyset, \\ false & if\ \mathcal{C}_\Sigma(\boldsymbol{X}') = \emptyset, \\ undefined & otherwise. \end{cases} \tag{2.24}$$

Example 2.3.4. *Given the QRC,*

$$(\forall p \in [2,3]')x + p \geq 0, \tag{2.25}$$

where $x \in [-1,2]'$. *By negating the previous QRC, the next QRC is obtained,*

$$\neg((\forall p \in [2,3]')x + p \geq 0) \Leftrightarrow (\exists p \in [2,3]')x + p < 0, \tag{2.26}$$

By applying a simple contractor over the negated QRC,

$$\mathcal{C}_{\neg\Sigma}([-1,2]') = [-1,2]' \cap ([-\infty,0]' - [2,3]') = \emptyset, \tag{2.27}$$

the empty set is obtained, which means that the original QRC is true.

Most of the contractors for solving QRCs are based on extended notions of *consistency techniques* like 2B-Consistency/3B-Consistency Lhomme (1993) and Box-Consistency Benhamou *et al.* (1999).

2.3.2.2 Approximate solution set

One way to approximate the solution set Σ of a bounded QRC consists of finding a box (interval vector) which is guaranteed to include Σ. Then, this box is an outer approximation of Σ. It is also possible to find a box which is guaranteed to be included into Σ. Then, this box is an inner approximation of Σ. However, this methodology can be especially inefficient depending on the topology of the solution set, as the obtained approximations can be either very overestimated or underestimated. This type of approximations can be efficiently achieved by means of techniques which allow to prune the search space from non-solution/solution parts like Interval Constraint Propagation. Figure 2.2 shows two boxes approximating the solution set of Example 2.2.1.

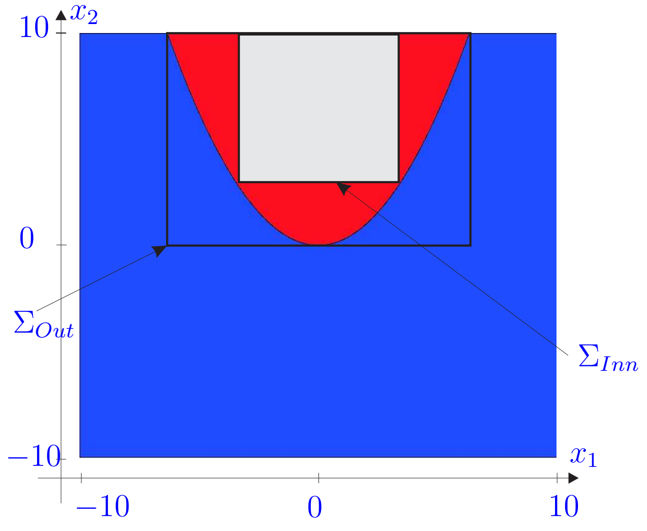


Figure 2.2: Box approximations for Σ of Example 2.2.1.

Another way to approximate a solution set is by means of a set of non-overlapping boxes, also referred to as a paving, obtained using a branch-and-bound algorithm. This last method provides better approximations of the solution set, although it suffers from a higher computational complexity limiting its applicability. Basically, these algorithms repeatedly bisect the free-variable space $\boldsymbol{X}'$ and test after each bisection, using a validated numerical technique, whether the QRC holds or not on the resulting box . If the QRC holds everywhere on the resulting box, it is not bisected anymore and thus denoted as a true box, meaning that $\boldsymbol{X}'$ is contained in Σ. On the other hand, if the QRC does not hold in any point of $\boldsymbol{X}'$, the box is not bisected anymore and denoted as a false box, meaning that it is contained in complementary set $\neg\Sigma$. If none of the tests holds, the box is denoted as undefined and it is bisected. This procedure is repeated until a predefined error bound ϵ is reached. Here the error bound can describe the fraction of the volume of the free-variable space for which the solution set membership should be decided. However, other criterions can be used. Thus, we can obtain an inner approximation of Σ (Σ_{Inn}) by means of the set of boxes which have been classified as true and an outer approximation Σ_{Out} by means of the set of true boxes together with the set of undefined boxes. It is important to remark that the use of branch-and-bound algorithms implies an exponential running time if one wants to come arbitrarily close to an exact solution. This exponential complexity is directly related to the number of free-variables to be bisected.

Figure 2.3 graphically shows the branch-and-bound procedure on a generic two dimensional problem and Algorithm 1 summarizes the main steps of this algorithm in a pseudocode form. Figure 2.4 shows the graphical output of a branch-and-bound based algorithm, which corresponds to the resolution of Example 2.2.1, where red boxes are included to the solution set Σ, blue boxes are included in the complementary set $\neg\Sigma$ and green boxes are undefined.

If a contractor is used as validated numerical technique, not only it can test the truth-value of a QRC on a given box $\boldsymbol{X}'$, but also can reduce the average runtime of the branch-and-bound algorithms. This is done by replacing expensive exhaustive search as much a possible, by methods for pruning elements from the search space for which it is easy to show that they not contain solutions. For this reason, algorithms based on branch-and-bound techniques and contractors are commonly called branch-and-prune algorithms.

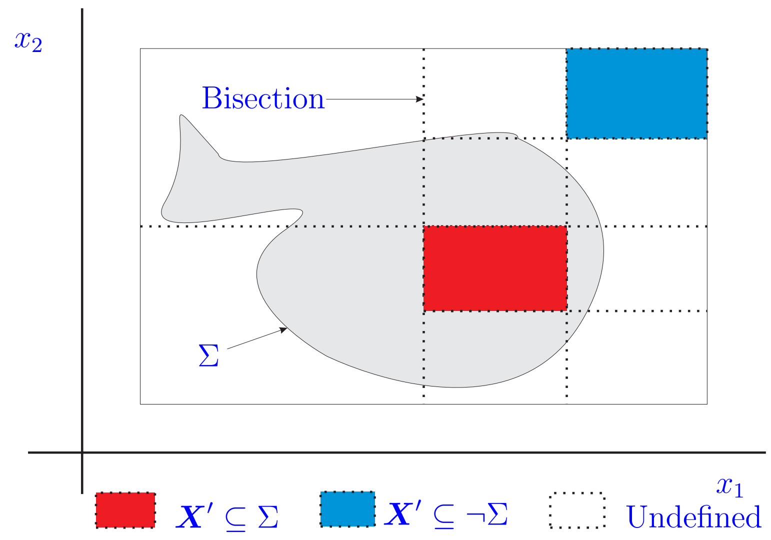

x_2
Bisection
Σ
x_1
$\boldsymbol{X}' \subseteq \Sigma$
$\boldsymbol{X}' \subseteq \neg\Sigma$
Undefined

Figure 2.3: Branch-and-bound algorithm.

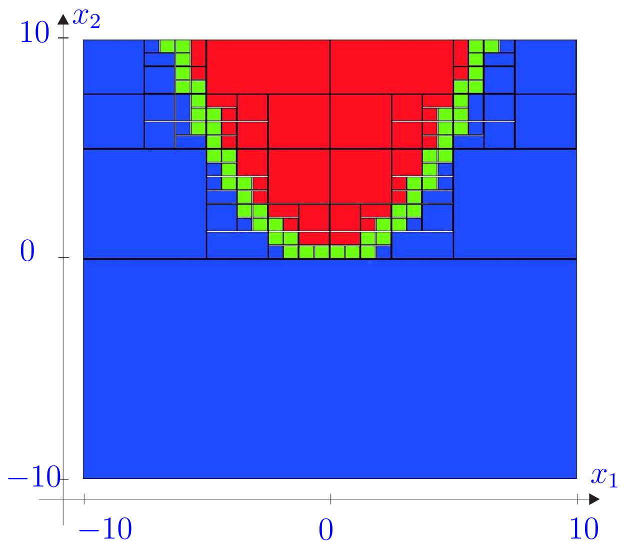


Figure 2.4: Paving for Σ of Example 2.2.1.

Algorithm 1 Branch-and-Bound Algorithm

```
Input: A QRC ($\phi$), $\boldsymbol{X'}$ and $\epsilon$.
Output: $\Sigma_{Inn}$ and $\Sigma_{Out}$ of the solution set of $\phi$.
  Enqueue $\boldsymbol{X'}$ to $ListBox$; $\Sigma_{Inn} := \emptyset$;
  while $ErrorBound > \epsilon$ do
    Dequeue $\boldsymbol{X'}$ from $ListBox$;
    if $\phi(\boldsymbol{X'})$ holds then
      Enqueue $\boldsymbol{X'}$ to $\Sigma_{Inn}$;
    else if $\phi(\boldsymbol{X'})$ does not hold then
      Do nothing;
    else
      Bisect $\boldsymbol{X'}$ and enqueue the resulting boxes to $ListBox$;
    end if
  end while
  Enqueue $\Sigma_{Inn}$ and $ListBox$ to $\Sigma_{Out}$;
  return $\Sigma_{Inn}$ and $\Sigma_{Out}$;
```

where

- $\boldsymbol{X'}$: Box or interval vector.
- $ListBox$: List of boxes.
- Σ_{Inn}: List of boxes such that $\Sigma_{Inn} \subseteq \Sigma$.
- Σ_{Out}: List of boxes such that $\Sigma \subseteq \Sigma_{Out}$.
- Enqueue: The result of adding a box to a list.
- Dequeue: The result of extracting a box from a list.
- $ErrorBound$: Percentage of undefined search space respect the initial search space.
- ϵ: A real value.

Moreover, negating the QRC, it is possible to build a contractor for the complementary solution set $\neg\Sigma$, which eliminates parts belonging to the solution set. Therefore, an inner contractor for Σ can also be obtained. Figure 2.5 graphically shows the branch-and-prune algorithm on a two-dimensional example.

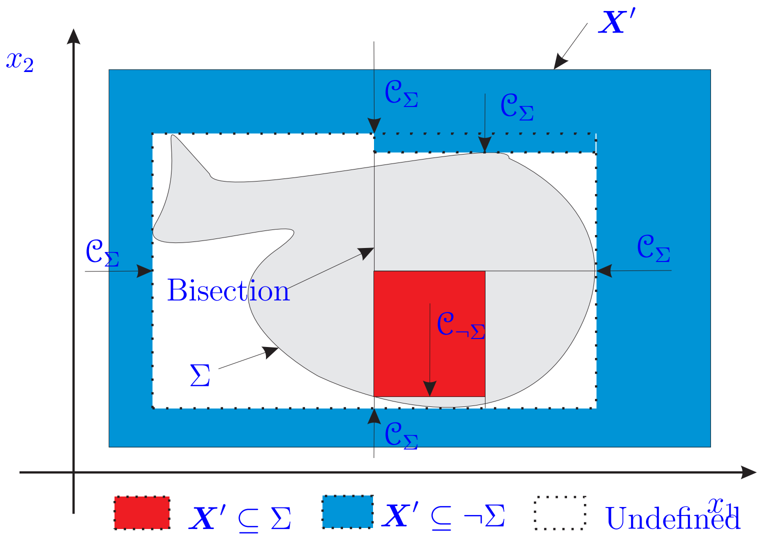


Figure 2.5: Branch-and-prune algorithm.

Complexity analysis and termination: The complexity of a branch-and-bound algorithm like Algorithm 1 is in general exponential in the problem dimension (number of free variables) Horowitz *et al.* (1997). However, it is not possible to give a more precise complexity limit because it depends of lots of factors like the shape, the length or the area of the solution set (Σ). On the other hand, the complexity of an algorithm implementing a contractor using constraint propagation is polynomial in the problem dimension Bordeaux *et al.* (2001); Collavizza *et al.* (1999). Therefore, using a contractor is efficient compared to expensive exhaustive search and should be used as much as possible.

Concerning the termination of Algorithm 1, it is important to see that the problem of computing truth-values/solution sets of QRCs can be numerically ill-posed Ratschan (2001). An example is the QRC,

$$(\exists x \in [-1,1]') - x^2 = 0, \tag{2.28}$$

which is true, but becomes false under arbitrarily small positive perturbations of the constant 0. As a consequence, it is not possible to design an algorithm based on approximation that will always terminate (with a correct result). Note that this situation is similar for most computational problems of continuous mathematics (e.g., solving linear equations, solving differential equations). However, as in these cases, most inputs are still numerically well-posed. One can even argue that, philosophically speaking, the well-posed problems are exactly the problems that model real-life problems in a meaningful way. It is beyond the scope of this thesis to present all the formal details for characterizing well-posed QRCs.

Next sections review some of the existing approximate methods for solving QRCs, highlighting their advantages and their limitations.

2.3.2.3 Cylindrical Box Decomposition

A singular approach, which combines QE methods with validated numerical techniques is the Hong's method Hong (1995). This approach provides a symbolic-numeric algorithm called Cylindrical Box Decomposition, which is an extension of the CAD algorithm Collins (1975) in combination with Interval Arithmetic Moore (1966). The Hong's method tries to deal with the more general problem, however, it is still too slow to be useful, cannot guarantee termination, and is too complicated to allow a thorough study of its properties. Some of these deficiencies have been removed by Ratschan, by giving an exact classification of the cases when the problem is numerically ill-posed Ratschan (2001), proposing to deal with numerically ill-posed problems by approximate quantifiers Ratschan (2003a), and providing a new algorithm Ratschan (2002b) with a quite efficient implementation called AQCS Solver Ratschan (2002a). Despite of being theoretically general, this methodology is only suitable for small-size problems and some middle-size problems due to its complexity.

2.3.2.4 Multivariate Bernstein Polynomials

Multivariate Bernstein Polynomials (MBPs) Garloff (1993) can be used to estimate the range of a polynomial function over a given domain. The idea behind MBPs consists on finding the coefficients of the so-called Bernstein Polynomials for a given polynomial function and the range of this last is obtained by means of the combination of these coefficients.

The main advantage of MBPs methods is that they can be computationally less expensive than other approximate methods (e.g. Interval Analysis) in some special cases. However, MBPs methods are restricted to problems involving strict polynomial inequalities and only accepts universal quantification over the variables.

Nevertheless, several applications in the Robust Control domain are found. For example, in Malan *et al.* (1992), Bernstein branch-and-bound methods are applied to robust performance analysis. In Zettler & Garloff (1998), improved Bernstein branch-and-bound methods are used for robust stability analysis. In Vicino *et al.* (1990) robust stability margin for multivariate polynomials systems is computed.

When the QRC involves predicates which are not polynomial, other validated numerical techniques, like Interval Analysis, are required.

2.3.2.5 Interval methods

Methods based on Interval Analysis can solve a more general class of QRCs than MBPs methods do, but they still have some limitations on the form of the QRCs (e.g. only one type of quantification, inequality predicates, linear predicates) and inherit an important drawback from Interval Analysis, the overestimation phenomenon due to the *dependency* and *wrapping* effects. Moreover, this approach remains only applicable to middle-size problems because of its computational complexity. Nevertheless, interval based methods have been widely applied to solve QRCs in different fields like Robust Control Jaulin & Walter (1996); Malan *et al.* (1997); Vehí *et al.* (1999, 2000) and Parameter Identification Jaulin & Walter (1993, 1999).

Several works have been proposed to overcome some of these limitations:

- Jaulin, with his well-known SIVIA (Set Inversion Via Interval Analysis) algorithm Jaulin & Walter (1993, 1999), proposed a methodology for solving a class of QRCs involving non-linear inequality constraints and only one type of quantification ($\forall$ or $\exists$) over the parameters. Several applications to Robust Control and Parameter Identification are proposed in his work.

- In order to tackle with the overestimation problem, Vehí proposed in his thesis Vehí (1998) the use of Modal Interval Analysis Gardeñes *et al.* (2001) to deal with the dependency effect. Applications to the problem of controllers design are also proposed in his work.

- When only linear predicates are involved, Shary Shary (2002) proposed an efficient technique, not based on bisection techniques, which allow to find an inner and an outer approximation of the solution set of a QRC by means of a single box. By using the so-called Kaucher Complete Interval Arithmetic Kaucher (1980) combined with different existence theorems, this technique extends the classical interval Gauss-Seidel algorithm Neumaier (1990) to tackle with QRCs involving AE-quantification (universal quantifier precedes the existential quantifier). A similar approach, based on Modal Interval Analysis, was proposed by Sainz in Sainz *et al.* (2002a,b). A more recent work by Goldsztejn Goldsztejn (2005), proposed some improvements to the work proposed by Shary, in order to obtain better approximations of the solution of the QRC by using parallelepiped boxes instead of square boxes.

- Goldsztejn proposed in Goldsztejn (2003) an algorithm which combines interval branch-and-bound techniques with a parametric version of the Miranda theorem Kearfott (2001). His approach allows to obtain inner and outer approximations of the solution set of a class of quantified constraints involving $\forall\exists$-quantification and nonlinear equality predicates not sharing existentially quantified variables.

2.3.2.6 Contractor methods

With the purpose of reducing the computational complexity of the interval branch-and-bound approaches, several authors have proposed techniques based on contractors for solving different instances of QRCs:

- In Benhamou & Goualard (2000), Benhamou et al. propose an algorithm, based on an extended notion of the Box-Consistency technique to obtain inner and outer approximations of the solution set of QRCs involving inequality nonlinear real predicates and only one universal quantified variable. An interesting application of this technique to the problem of camera control is presented in Benhamou *et al.* (2004).

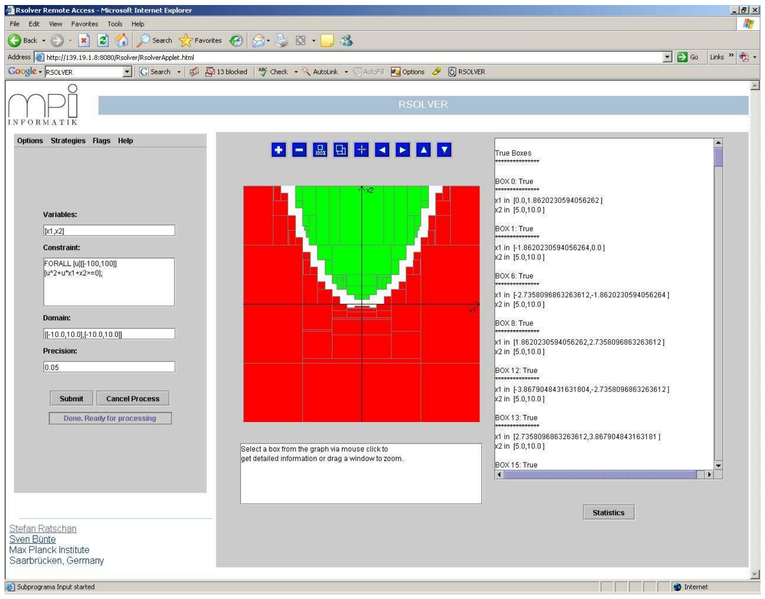
- A more general algorithm for solving QRCs involving inequality constraints and only one type of quantification ($\forall$ or $\exists$) is proposed by Braems et al. in Jaulin *et al.* (2002) by means of an algorithm referred to as the Projection Algorithm. This approach, based on Forward/Backward Propagation and optimization techniques, has a practical implementation, called Proj2D and available at Dao (2005). Applications based on this technique have been proposed to solve different automation problems Dao *et al.* (2003); Jaulin *et al.* (2002).

- More recently, Ratschan has proposed a more general framework Ratschan (2002c, 2003b) which is based on a extended version of a consistency technique for first-order constraints. This technique can deal with AE-quantification and theoretically accepts one equality and several inequalities. A practical useful implementation, called RSOLVER, is available at Ratschan (2005).

- One of the most recent works is the one of Goldsztejn Goldsztejn & Jaulin (2005), which proposes a bisection technique combined with a contractor based on a generalized interval parametric Hansen-Sengupta operator Hansen & Sengupta (1981). This approach, partially solves a well known open problem in set computation consisting on computing the inner approximation of the range of vector-valued function. This problem is equivalent to that of computing the inner approximation of the solution set of a QRC with equality predicates sharing existentially quantified variables. However, the methodology is restricted to problems with the same number of equality predicates than existentially quantified variables and does not guarantee termination.

As mentioned before, contractors can notably reduce the average run-time of the algorithms for solving QRCs. However, current implementations are still far from being useful for solving big-size problems. Thus, further research in that direction is still being done.

Figure 2.6 and 2.7 shows the screenshots of two current existing solvers based on branch-and-prune techniques. These screenshots correspond to the graphical output for the QRC from Example 2.2.1.



Variables:
[x1,x2]

Constraint:
FORALL [u][[-100,100]]
[u^2+u*x1+x2>=0];

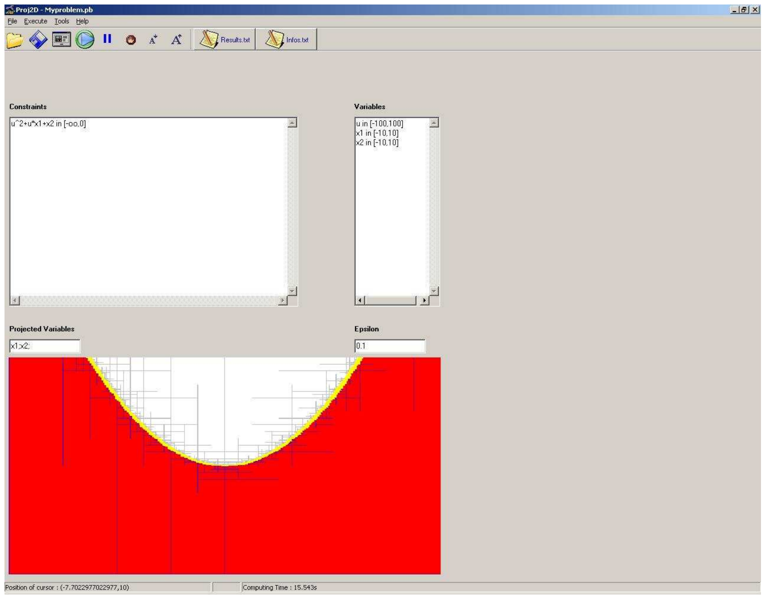
Domain:
[[-10.0,10.0],[-10.0,10.0]]

Precision:
0.05

Select a box from the graph via mouse click to get detailed information or drag a window to zoom.

```
True Boxes
***************

BOX 0: True
***************
x1 in  [0.0,1.8620230594056262 ]
x2 in  [5.0,10.0 ]

BOX 1: True
***************
x1 in  [-1.8620230594056264,0.0 ]
x2 in  [5.0,10.0 ]

BOX 6: True
***************
x1 in  [-2.7358096863263612,-1.8620230594056264 ]
x2 in  [5.0,10.0 ]

BOX 8: True
***************
x1 in  [1.8620230594056262,2.7358096863263612 ]
x2 in  [5.0,10.0 ]

BOX 12: True
***************
x1 in  [-3.8679048431631804,-2.7358096863263612 ]
x2 in  [5.0,10.0 ]

BOX 13: True
***************
x1 in  [2.7358096863263612,3.867904843163181 ]
x2 in  [5.0,10.0 ]

BOX 15: True
```

Figure 2.6: RSOLVER screenshot corresponding to Example 2.2.1.



Constraints

```
u^2+u*x1+x2 in [-oo,0]
```

Variables

```
u in [-100,100]
x1 in [-10,10]
x2 in [-10,10]
```

Projected Variables: x1;x2;

Epsilon: 0.1

```
Position of cursor : (-7.7022977022977,10)    Computing Time : 15.543s
```

Figure 2.7: Proj2D screenshot corresponding to Example 2.2.1.

2.4 Conclusions

This chapter states the class of quantified real constraint (QRC) to be solved in this thesis and reviews the principal existing techniques for solving such a problem, highlighting their main advantages and drawbacks. Solving big-size problems QRCs, still remains an open problem due to the computational complexity of the current approaches. Research on new techniques for reducing this complexity and for solving more general class of QRCs are the objectives of many research works. Next chapter, presents a new approach based on Modal Interval Analysis which intends to contribute to overcoming some of the existing limitations.

Chapter 3

Quantified Real Constraint Satisfaction Using Modal Intervals

This chapter deals with the satisfaction of the class of quantified real constraints (QRCs) stated in Section 2.2.2. An extended notion of the so-called *interval inclusion test* in interval analysis is presented, which enlarges the class of QRCs that can be dealt with respect to the classic Interval Analysis approach. This new notion of inclusion test is based on the theory of Modal Interval Analysis (MIA) and consequently it has been referred as to *modal interval inclusion test*. This chapter also presents an efficient algorithm, based on branch-and-bound techniques and MIA, to compute approximations of the *semantic extensions* of a continuous function, the modal interval counterparts of the *inclusion functions* in classic interval analysis, and the key tool for applying the *modal interval inclusion test*.

3.1 Introduction

As already introduced in Section 2.3.2.1, one way of proving the satisfaction of a QRC is by means of the interval analysis Moore (1966), and more specifically through the notion of *interval inclusion test*.

Remark 3.1.1. *Proving the consistency (truth-value) of a QRC (ϕ) is equivalent* *to answering the following question: Is ϕ true for whatever the values of the free-variables $\boldsymbol{x}$ on a given box $\boldsymbol{X}'$?*

■

Given a box $\boldsymbol{X}'$, an *interval inclusion test* can prove the consistency of a QRC of the form

$$(\forall \boldsymbol{u} \in \boldsymbol{U}') f(\boldsymbol{x}, \boldsymbol{u}) \ \geq 0, \tag{3.1}$$

where $\boldsymbol{u}$ is a vector of universally quantified parameters ranging on a box $\boldsymbol{U}'$, and f is a continuous real function in $\mathbb{R}^n \rightarrow \mathbb{R}$. However, it cannot prove, for instance, a QRC of the form,

$$(\exists \boldsymbol{v} \in \boldsymbol{V}') f(\boldsymbol{x}, \boldsymbol{v}) = 0, \tag{3.2}$$

where $\boldsymbol{v}$ is a vector of existentially quantified parameters.

Modal Interval Analysis (MIA) Gardeñes *et al.* (2001), can tackle with the previous logical statement and in general can deal with QRCs of the form,

$$(\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{V}') f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = 0, \tag{3.3}$$

where $\boldsymbol{u}$ and $\boldsymbol{v}$ are respectively vectors of parameters affected by the universal and existential quantifiers, $\boldsymbol{U}'$ and $\boldsymbol{V}'$ are real intervals and $f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v})$ is a continuous real function.

Remark 3.1.2. *Notice that any instance of the problem stated in Section 2.2.2, can be expressed as Equation 3.3 or multiple instances of the same problem. For example:*

- *Inequality predicates can be expressed as Equation 3.3 by simply introducing slack variables. For instance, $f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) \leq 0 \Leftrightarrow f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) - a = 0$, where $a \in [-\inf, 0]'$ is a slack variable.*

- *Solving several predicates, not sharing existentially quantified variables, is equivalent to solving multiple instances of Equation 3.3. For example,*

$$\begin{aligned}&(\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v}_1 \in \boldsymbol{V}'_1)\ldots(\exists \boldsymbol{v}_n \in \boldsymbol{V}'_n)\\ &(g_1(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}_1)=0 \wedge \ldots \wedge g_n(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}_n)=0)\\ &\Leftrightarrow (\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v}_1 \in \boldsymbol{V}'_1)g_1(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}_1)=0 \wedge \ldots\\ &\wedge(\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v}_n \in \boldsymbol{V}'_n)g_n(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}_n)=0. \qquad (3.4)\end{aligned}$$

- *Logical conjunctions of inequality predicates sharing existentially quantified variables, can be reduced to a single predicate using the interval min (or max) function defined by*

$$min([\underline{a},\overline{a}],[\underline{b},\overline{b}]) := [min(\underline{a},\underline{b}), min(\overline{a},\overline{b})]. \qquad (3.5)$$

  For example, for inequality predicates of the form <, the next equivalence can be used,

$$\begin{aligned}&(\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v}_1 \in \boldsymbol{V}'_1)\ldots(\exists \boldsymbol{v}_n \in \boldsymbol{V}'_n)\\ &(g_1(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v})<0 \wedge \ldots \wedge g_n(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v})<0)\\ &\Leftrightarrow (\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v}_1 \in \boldsymbol{V}'_1)\ldots(\exists \boldsymbol{v}_n \in \boldsymbol{V}'_n)\\ &min(g_1(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}),\ldots,g_n(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}))<0. \qquad (3.6)\end{aligned}$$

This way of solving inequality predicates sharing existentially quantified variables is an original contribution of this thesis.

■

Notice that proving the satisfaction of Equation 3.3 on a given box $\boldsymbol{X}'$, is equivalent to looking for the truth-value of the next first-order logic formula,

$$(\forall \boldsymbol{x} \in \boldsymbol{X}')(\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{V}')f(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}) \ = 0, \qquad (3.7)$$

something that can be naturally done by MIA. Next section is devoted to the satisfaction of the previous logical formula using the theory of MIA, and more specifically through the notion of the *modal interval inclusions test*.

3.2 Modal Interval Analysis

This Section presents some basic concepts and results about MIA for the understanding of thesis. For more details and proofs the reader can see Gardeñes *et al.* (2001); SIGLA/X (1999). In a second part, the notion of *modal interval inclusion test* is introduced, which provides the key tool for the resolution of the problem stated in Equation 3.3.

3.2.1 Basic concepts

Remark 3.2.1. *In order to distinguish a modal interval from a classic interval, the classic interval has been denoted with a prime mark (e.g. A').*

■

Unlike a classical interval $A' = [\underline{a}, \overline{a}]'$, with $\underline{a} \leq \overline{a}$, which is a set of real numbers

$$A' = [\underline{a}, \overline{a}]' := \{x \in \mathbb{R} \mid \underline{a} \leq x \leq \overline{a}\}, \tag{3.8}$$

a modal interval is a pair formed by a classical interval and a quantifier. Therefore there exist two types of modal intervals: $A := (A', \forall)$ named *improper* intervals and represented by $A = [\underline{a}, \overline{a}]$, with $\underline{a} \geq \overline{a}$, and $B := (B', \exists)$ named *proper* intervals and represented by $B = [\underline{b}, \overline{b}]$, with $\underline{b} \leq \overline{b}$. Notice that proper intervals are identifiable with intervals from the classic Interval Analysis theory. The set of modal intervals is represented by $I^*(\mathbb{R})$.

Example 3.2.1. *For example, the modal interval* $[2, 5]$ *is equal to* $([2, 5]', \exists)$*, and the modal interval* $[8, 4]$ *is equal to* $([4, 8]', \forall)$.

■

For both type of modal intervals, the bounds $\underline{a}$ and $\overline{a}$ are referred to as the *infimum*, $\underline{a} = Inf([\underline{a}, \overline{a}])$ and the *supremum*, $\overline{a} = Sup([\underline{a}, \overline{a}])$. A *point-wise* interval $[a, a]$, also represented as $[a]$, can be considered as proper or improper and it is identifiable with the real number a.

For an interval $A = [\underline{a}, \overline{a}]$, the operators *Prop*, *Impr* and *Dual* are defined as

$$Prop([\underline{a}, \overline{a}]) := \begin{cases} [\underline{a}, \overline{a}] & \text{if } \underline{a} \leq \overline{a} \\ [\overline{a}, \underline{a}] & \text{if } \underline{a} > \overline{a}. \end{cases} \tag{3.9}$$

$$Impr([\underline{a}, \overline{a}]) := \begin{cases} [\overline{a}, \underline{a}] & \text{if } \underline{a} \leq \overline{a} \\ [\underline{a}, \overline{a}] & \text{if } \underline{a} > \overline{a}. \end{cases} \tag{3.10}$$

$$Dual([\underline{a}, \overline{a}]) := [\overline{a}, \underline{a}]. \tag{3.11}$$

Example 3.2.2.

$$Prop([5, -5]) = [-5, 5],$$

$$Impr([-5, 5]) = [5, -5],$$

$$Dual([-5, 5]) = [5, -5].$$

■

3.2.1.1 Modal interval inclusion

Unlike a classical interval, which is identified by a set of real numbers, a modal interval $A := (A', Q_A)$ is identified by the set of real predicates accepted by itself:

$$Pred((A', Q_A)) := \{P(.) \in Pred(\mathbb{R}) \mid (Q_A x \in A')\ P(x)\}. \tag{3.12}$$

Example 3.2.3. *Given the modal interval $A = ([-1, 1]', \exists)$, the real predicate $P(.) = x > 0$ is accepted by the interval A because, $(\exists x \in [-1, 1]')x > 0$ is true. However, for the modal interval $B = ([-1, 1]', \forall)$, the same real predicate is not accepted because $(\forall x \in [-1, 1]')x > 0$ is false.*

■

With the identification of a modal interval with the set of those real predicates that it accepts ($X \leftrightarrow P(X)$), arises the inclusion of two intervals as the inclusion of the set of predicates that they accept; that is, if $A, B \in I^*(\mathbb{R})$

$$A \subseteq B \Leftrightarrow Pred(A) \subseteq Pred(B). \tag{3.13}$$

The inclusion between two modal interval maintains the same *modus operandi* that its classic interval counterpart.

$$[\underline{a}, \overline{a}] \subseteq [\underline{b}, \overline{b}] \Leftrightarrow (\underline{a} \geq \underline{b} \wedge \overline{a} \leq \overline{b}). \tag{3.14}$$

Example 3.2.4. *Given two modal interval* $A = ([-1,1]', \exists) = [-1,1]$ *and* $B = ([-1,1]', \forall) = [1,-1]$*,* B *is included in* A *because* $[1,-1] \subseteq [-1,1]$ *is true, which means that the set of accepted predicates by* B *is included to the set accepted by* A*.*

■

3.2.1.2 Modal interval lattice operators

The interval lattice operators *meet* ($\vee$) and *join* ($\wedge$) on $I^*(\mathbb{R})$ for a bounded family of modal intervals $A(I) := \{A(i) = [a_1(i), a_2(i)] \in I^*(\mathbb{R}) \mid i \in I\}$ (I is the index's domain) are defined as the $\subseteq$-maximum interval contained in all $A(i)$, for the meet, and the $\subseteq$-minimum interval that contains all $A(i)$, for the join; i.e.,

$$\bigwedge_{i \in I} A(i) = A \in I^*(\mathbb{R}) \text{ is such that } (\forall i \in I)\ X \subseteq A(i) \Leftrightarrow X \subseteq A, \tag{3.15}$$

$$\bigvee_{i \in I} A(i) = B \in I^*(\mathbb{R}) \text{ is such that } (\forall i \in I)\ X \supseteq A(i) \Leftrightarrow X \supseteq B, \tag{3.16}$$

denoted by $A \wedge B$ and $A \vee B$ for the corresponding two-operands case. The result, as a function of the interval bounds, is

$$\bigwedge_{i \in I} A(i) = [\max_{i \in I} a_1(i), \min_{i \in I} a_2(i)] \tag{3.17}$$

$$\bigvee_{i \in I} A(i) = [\min_{i \in I} a_1(i), \max_{i \in I} a_2(i)]. \tag{3.18}$$

Example 3.2.5. *Given two modal interval $A = [-1, 1]$ and $B = [8, 6]$,*

$$A \wedge B = [-1, 1] \wedge [8, 6] = [8, 1],$$

$$A \vee B = [-1, 1] \vee [8, 6] = [-1, 6].$$

■

3.2.1.3 Modal interval arithmetic

The modal interval arithmetic coincides with the so-called Kaucher Complete Interval Arithmetic Kaucher (1980). However, MIA not only extends the classic interval arithmetic to the whole interval lattice but, it provides a logical meaning to the results, which is related to the modality of the involved intervals (e.g. proper or improper).

Example 3.2.6. *Given two interval $A = [\underline{a}, \overline{a}]$ and $B = [\underline{b}, \overline{b}]$, the basic operations $+$ and $-$ as a function of the interval bounds are:*

$$Sum: \; A + B = [\underline{a} + \underline{b}, \overline{a} + \overline{b}]. \tag{3.19}$$

$$Rest: \; A - B = [\underline{a} - \overline{b}, \overline{a} - \underline{b}]. \tag{3.20}$$

Then, for $A = [-1, 1]$ and $B = [1, -1]$,

$$A + B = [-1 + 1, 1 + (-1)] = [0, 0].$$

$$A - B = [-1 - (-1), 1 - 1] = [0, 0].$$

■

3.2.1.4 *-Semantic extension

A key concept in MIA is the *-semantic extension (f^*) of a continuous function f to a modal interval vector $\boldsymbol{X} \in I^*(\mathbb{R}^n)$, which can be seen as the modal interval counterpart of the range (or interval united extension) of a continuous function in classic Interval Analysis. f^* is defined by

$$\begin{aligned} f^*(\boldsymbol{X}) &:= \bigvee_{\boldsymbol{u}\in\boldsymbol{U}'} \bigwedge_{\boldsymbol{v}\in\boldsymbol{V}'} [f(\boldsymbol{u},\boldsymbol{v}), f(\boldsymbol{u},\boldsymbol{v})] = \\ &= [\min_{\boldsymbol{u}\in\boldsymbol{U}'} \max_{\boldsymbol{v}\in\boldsymbol{V}'} f(\boldsymbol{u},\boldsymbol{v}), \max_{\boldsymbol{u}\in\boldsymbol{U}'} \min_{\boldsymbol{v}\in\boldsymbol{V}'} f(\boldsymbol{u},\boldsymbol{v})], \end{aligned} \tag{3.21}$$

where $\boldsymbol{x} = (\boldsymbol{u},\boldsymbol{v})$ is the component-splitting corresponding to the proper and improper components of $\boldsymbol{X} = (\boldsymbol{U},\boldsymbol{V})$.

Remark 3.2.2. *Notice that if $\boldsymbol{X}$ has only proper components,*

$$f^*(\boldsymbol{X}) := \bigvee_{\boldsymbol{x}\in\boldsymbol{X}'} [f(\boldsymbol{x}), f(\boldsymbol{x})] = [\min_{\boldsymbol{x}\in\boldsymbol{X}'} f(\boldsymbol{x}), \max_{\boldsymbol{x}\in\boldsymbol{X}'} f(\boldsymbol{x})], \tag{3.22}$$

it corresponds to the interval united extension R_f of the classic interval analysis and is the range of f in the parameter space $\boldsymbol{X}'$.

■

Another semantic extension defined in MIA is the **-semantic extension, which is the dual formulation of the *-semantic extension. It is defined by

$$\begin{aligned} f^{**}(\boldsymbol{X}) &:= \bigwedge_{\boldsymbol{u}\in\boldsymbol{U}'} \bigvee_{\boldsymbol{v}\in\boldsymbol{V}'} [f(\boldsymbol{u},\boldsymbol{v}), f(\boldsymbol{u},\boldsymbol{v})] = \\ &= [\max_{\boldsymbol{u}\in\boldsymbol{U}'} \min_{\boldsymbol{v}\in\boldsymbol{V}'} f(\boldsymbol{u},\boldsymbol{v}), \min_{\boldsymbol{u}\in\boldsymbol{U}'} \max_{\boldsymbol{v}\in\boldsymbol{V}'} f(\boldsymbol{u},\boldsymbol{v})]. \end{aligned} \tag{3.23}$$

The f^{**} can be easily obtained through the next relation

$$f^{**}(\boldsymbol{X}) = Dual(f^*(Dual(\boldsymbol{X}))), \tag{3.24}$$

which makes the implementation of f^{**} unnecessary.

In the special case when $f^*(\boldsymbol{X}) = f^{**}(\boldsymbol{X})$, f is said to be *JM-commutable* (Join Meet commutable) for $X \in I^*(\mathbb{R}^n)$. Important examples of JM-commutable functions are the one-variable continuous functions and every two-variable continuous function $f(x,y)$ that is partially monotonic in a domain (X', Y'), like the arithmetic operators $x+y$, $x-y$, $x*y$, x/y and others like x^y, $\max(x,y)$ and $\min(x,y)$. Thus, their semantic extensions can be computed using simple arithmetic computations with the interval bounds.

3.2.1.5 *-Semantic theorem

A very important result about the *-semantic extension is the so-called *-semantic theorem:

Theorem 3.2.1 ($*-$semantic theorem)**.** *Let $Z \in I^*(\mathbb{R})$ be a modal interval. Then*

$$f^*(\boldsymbol{U},\boldsymbol{V}) \subseteq Z \Leftrightarrow (\forall \boldsymbol{u} \in \boldsymbol{U}')(Qz \in Z')(\exists \boldsymbol{v} \in \boldsymbol{V}')z = f(\boldsymbol{u},\boldsymbol{v}), \tag{3.25}$$

where $Q = \exists$ when Z is a proper interval and $Q = \forall$ when Z is an improper interval.

■

Theorem 3.2.1 states an equivalence between a first-order logic formula, involving equalities relating to a continuous real function, and an interval inclusion. Moreover, Theorem 3.2.1 can be extended to the following implication when the interval inclusion is fulfilled by an outer approximation of the *-semantic extension,

$$Outer(f^*(\boldsymbol{U},\boldsymbol{V})) \subseteq Z \Rightarrow (\forall \boldsymbol{u} \in \boldsymbol{U}')(Qz \in Z)(\exists \boldsymbol{v} \in \boldsymbol{V}')z = f(\boldsymbol{u},\boldsymbol{v}), \tag{3.26}$$

where $Outer(f^*(\boldsymbol{U},\boldsymbol{V}))$ is an outer approximation of f^*.

When the functional relation is $f(\boldsymbol{u},\boldsymbol{v}) = 0$, it is enough to put $Z = [0,0]$ and to omit the term $(Qz \in Z')$, which exactly leads to the resolution of the problem stated in Equation 3.7.

In the case of the $\geq$ or $\leq$ relations, the *-semantic theorem states the following equivalences:

$$f^*(\boldsymbol{U},\boldsymbol{V}) \subseteq (-\infty,0] \;\Leftrightarrow\; (\forall \boldsymbol{u} \in \boldsymbol{U}')\;(\exists \boldsymbol{v} \in \boldsymbol{V}')\; f(\boldsymbol{u},\boldsymbol{v}) \leq 0, \tag{3.27}$$

and

$$f^*(\boldsymbol{U},\boldsymbol{V}) \subseteq [0,+\infty) \;\Leftrightarrow\; (\forall \boldsymbol{u} \in \boldsymbol{U}')\;(\exists \boldsymbol{v} \in \boldsymbol{V}')\; f(\boldsymbol{u},\boldsymbol{v}) \geq 0, \tag{3.28}$$

Remark 3.2.3. *Notice that inequality predicates can also be expressed as equality predicates by introducing slack variables, for example,*

$$f(u,v) \leq 0 \Leftrightarrow f(u,v) - a = 0, \tag{3.29}$$

where $a \in [-\inf, 0]$ *is a slack variable.*

■

When the negation of the logical formula from Equation 3.7 is considered, in accordance with the *-semantic theorem

$$\begin{aligned} f^*(\boldsymbol{U},\boldsymbol{V}) \nsubseteq [0,0] \quad &\Leftrightarrow \quad \neg((\forall \boldsymbol{u} \in \boldsymbol{U}') \ \ (\exists \boldsymbol{v} \in \boldsymbol{V}') \ f(\boldsymbol{u},\boldsymbol{v}) = 0) \\ &\Leftrightarrow \quad (\exists \boldsymbol{u} \in \boldsymbol{U}') \ (\forall \boldsymbol{v} \in \boldsymbol{V}') \ f(\boldsymbol{u},\boldsymbol{v}) \neq 0 \\ &\Rightarrow \quad (\forall \boldsymbol{v} \in \boldsymbol{V}') \ (\exists \boldsymbol{u} \in \boldsymbol{U}') \ f(\boldsymbol{u},\boldsymbol{v}) \neq 0. \end{aligned} \tag{3.30}$$

Therefore, the negation the logical formula stated in Equation 3.3 can be verified proving a modal interval non-inclusion. If an inner approximation of f^* is considered, the following assertion is also verified

$$Inn(f^*(\boldsymbol{U},\boldsymbol{V})) \nsubseteq [0,0] \Rightarrow \ (\forall \boldsymbol{v} \in \boldsymbol{V}') \ (\exists \boldsymbol{u} \in \boldsymbol{U}') \ f(\boldsymbol{u},\boldsymbol{v}) \neq 0. \tag{3.31}$$

It has been shown how the satisfaction of the logical statement expressed in Equation 3.3 can be done by means of an interval inclusion involving f^*. In other words, the problem has been transferred to the computation of the *-semantic extension of a continuous function.

3.2.1.6 *-Semantic extension computation

Sometimes, under certain conditions, the computation of the *-semantic extension of a continuous function f can be easily done through simple modal interval computations by simply replacing, in the syntactic tree of f, its real variables ($\boldsymbol{x}$) by modal interval variables ($\boldsymbol{X}$) and the real arithmetic operators by their corresponding modal interval counterparts. This interval extension of a continuous function f is referred to as *modal interval rational extension* fR.

However, the modal interval rational extension, which can be obtained by these transformations, is in general not interpretable. This lack of interpretability is equivalent to saying that the inclusion

$$f^*(\boldsymbol{X}) \subseteq fR(\boldsymbol{X}). \tag{3.32}$$

is not satisfied and consequently the *-semantic theorem cannot be applied.

Example 3.2.7. *Let $f(x_1, x_2) = x_1 + x_2$ be a continuous function. Replacing the real variables (x_1, x_2) by the intervals $(X_1, X_2) = ([-5, 5], [6, -6])$ and the real operators by their intervals counterparts, the following result is obtained*

$$f^*(\boldsymbol{X}) = [-5, 5] + [6, -6] = [1, -1]. \tag{3.33}$$

According to Theorem 3.2.1, the semantic of the previous computation is,

$$(\forall x_1 \in [-5, 5])(\forall z \in [-1, 1])(\exists x_2 \in [-6, 6])\ x_1 + x_2 = z. \tag{3.34}$$

*However, the *-semantic extension of a function like $f(x_1, x_2) = x_1 * x_2 + x_2^2$, cannot be obtained by simply replacing the interval variables and operators. Its rational extension to the same interval $\boldsymbol{X} = ([-5, 5], [6, -6])$ is*

$$fR(\boldsymbol{X}) = [-5, 5] * [6, -6] + [6, -6]^2 = [360, 0], \tag{3.35}$$

but this result is not interpretable because $f^*(\boldsymbol{X}) = [6, 0] \nsubseteq [360, 0]$.

■

Definition 3.2.1. *A rational computation fR is called* optimal *on $\boldsymbol{X}$ when*

$$f^*(\boldsymbol{X}) = fR(\boldsymbol{X}) = f^{**}(\boldsymbol{X}), \tag{3.36}$$

considering an exact interval arithmetic. However, this optimality is rarely satisfied.

■

3.2.1.7 Interpretable modal interval rational extension

MIA provides a set of results which allow to guarantee an inclusion of the kind $f^*(\boldsymbol{X}) \subseteq fR(\boldsymbol{X})$ and consequently to obtain the desired interpretability given by the *-semantic theorem. Several theorems provide different results which allow to obtain interpretable modal interval rational extensions. The next definitions are required for a correct understanding of the following results (for a detailed description of these theorems see Gardeñes *et al.* (2001)).

Definition 3.2.2. *An* incidence *is an occurrence of a variable inside a function. For instance, given the interval function $F(X) = X * X$, the variable X has two incidences.*

■

Definition 3.2.3. *A variable is* multi-incident *when multiple incidences of itself appear inside a function. For instance, given the interval function $F(X) = X*X$, the variable X is said to be multi-incident. Therefore, an* uni-incident *variable is a variable that only occurs once.*

■

Definition 3.2.4. *The* monotony sense *of a real function f with respect to a variable x (or an incidence) over a given domain X' is defined by*

$$Monotony\ sense = \begin{cases} Increasingly\ monotonic\ if\ (\forall x \in X')\ \partial f/\partial x \geq 0, \\ Decreasingly\ monotonic\ if\ (\forall x \in X')\ \partial f/\partial x \leq 0, \\ Nonmonotonic\ otherwise. \end{cases} \tag{3.37}$$

This monotony-sense *can be calculated by evaluating the *-semantic extension of*

$d_x = \partial f/\partial x$ on $\boldsymbol{X}'$. Thus,

$$Monotony\ sense = \begin{cases} Increasingly\ monotonic\ if\ d_x^*(Prop(\boldsymbol{X})) \geq [0,0], \\ Decreasingly\ monotonic\ if\ d_x^*(Prop(\boldsymbol{X})) \leq [0,0], \\ Nonmonotonic\ otherwise. \end{cases} \tag{3.38}$$

■

Definition 3.2.5. *A real function f is called* totally monotonic *with respect to a multi-incident variable $x \in X'$ if it is uniformly monotonic for this variable and for each one of its incidences, considered as different variables.*

■

Definition 3.2.6. *A syntactic tree of a continuous function f is* tree-optimal *on $\boldsymbol{X}$ if, for any of its non-uniformly monotonic operators (e.g. $*, /$) it is followed downwards in the syntactic tree only by one-variable operators (e.g. pow, exp, sin) and upwards by uniformly monotonic operators (e.g. $+, -$). For instance, the continuous function*

$$f(x, y, z, u) = x * y + z * u, \tag{3.39}$$

which has a rational syntactic extension given by

$$fR(X, Y, Z, U) = X * Y + Z * U, \tag{3.40}$$

is tree-optimal in any $\boldsymbol{X} = (X, Y, Z, U) \in I^*(\mathbb{R}^4)$ because the involved non-uniformly monotonic operator $*$ is followed downward by variables. However, the continuous function

$$f(x, y, z, u) = (x + y) * (z + u), \tag{3.41}$$

which has rational syntactic extension given by

$$fR(X, Y, Z, U) = (X + Y) * (Z + U), \tag{3.42}$$

is not tree-optimal in all $I^*(\mathbb{R}^4)$ *because the non-uniformly monotonic operator* $*$ *is followed downward by the binary operator* $+$*. However, it can be optimal on a given* $\boldsymbol{X}$*. For instance, if* $0 \notin \boldsymbol{X}'$*,* fR *is optimal.*

■

The main results for outer approximating $f^*(\boldsymbol{X})$ can mainly be summarized by the next inclusion

$$f^*(\boldsymbol{X}) \subseteq OutR(fR(\boldsymbol{U}D, \boldsymbol{V}Dt)), \tag{3.43}$$

where D is the interval transformation over every *multi-incident* interval variable X_i such that, the continuous function f is *totally monotonic* with respect to $x_i \in X_i$, and consist of applying the duality operator over each *incidence* of X_i such that the *monotony-sense* of f with respect the incidence is contrary to the *monotony-sense* f with respect to the variable. t is the interval transformation corresponding to transform the *multi-incident* improper intervals $\boldsymbol{V}$ that are not *totally monotonic* into point-wise intervals defined for any of their points. $OutR$ is an outer rounding operator when floating-point operations Kahan (1996) are involved. When *uni-incident* improper interval variables are involved in $\boldsymbol{X}$, the t-transformation is only applied on these variables if the rational syntactic extension of f is not *tree-optimal*.

Example 3.2.8. *Given the real function* $f(u, v, z) = 2uv - uv + cos(z)$*. Let us compute an outer approximation of the *-semantic extension* $f^*([-1, 1], [2, 1], [10, -10])$ *by means of an interpretable rational interval function* $fR([-1, 1], [2, 1], [10, -10])$*. This function can be written as* $f(u, v, z) = 2u_1v_1 - u_2v_2 + cos(z)$*, where the subindexes represent the different multi-incidences. First of all, the monotony* *of f with respect to each variable and with respect to each one of its incidences, considered as different variables, needs to be computed.*

$$\begin{aligned}
&\partial f(u,v,z)/\partial u = 2v - v = v = [1,2] \geq 0,\\
&\partial f(u,v,z)/\partial u_1 = 2v \in 2 * [1,2]' = [2,4]' \geq 0,\\
&\partial f(u,v,z)/\partial u_2 = -v \in -[1,2]' = [-2,-1] \leq 0,\\
&\partial f(u,v,z)/\partial v = 2u - u = u = [-1,1] \supseteq 0,\\
&\partial f(u,v,z)/\partial v_1 = 2u \in 2 * [-1,1]' = [-2,2]' \supseteq 0,\\
&\partial f(u,v,z)/\partial v_2 = -u \in -[-1,1]' = [-1,1]' \supseteq 0,\\
&\partial f(u,v,z)/\partial z = -sin(z) \in -sin([-10,10]') = [-1,1]' \supseteq 0. \qquad (3.44)
\end{aligned}$$

Then, applying the corresponding D and t transformations, the next approximation is obtained

$$\begin{aligned}
&fR(UD, Vt, Z) = \qquad (3.45)\\
&2 * U_1 * Point(V_1) + Dual(U_2) * Point(V_2) + cos(Z) =\\
&2 * [-1,1] * [1.5,1.5] + [1,-1] * [1.5,1.5] + cos([10,-10]) = [-0.5,0.5],
\end{aligned}$$

where $Point$ is an operator returning a point-wise interval corresponding to any point of an interval (e.g. the center). Notice that the obtained result is an outer approximation of the exact result $f^*([-1,1],[2,1],[10,-10]) = [0,0]$.

■

MIA also provides results for inner approximating of the *-semantic extension of a continuous function, which can be mainly summarized by the next inclusion

$$InnR(fR(\boldsymbol{U}Dt, \boldsymbol{V}D)) \subseteq f^*(\boldsymbol{x}), \qquad (3.46)$$

where $InnR$ is an inner rounding operator when floating-point operations Kahan (1996) are involved.

Example 3.2.9. *Consider the same problem of Example 3.2.8 but now, let us compute an inner approximation of* $f^*([-1,1],[2,1],[10,-10])$,

$$\begin{aligned}
&fR(UD, Vt, Z) = \\
&2 * U_1 * V_1 + Dual(U_2) * V_2 + cos(Z) = \\
&2 * [-1,1] * [2,1] + [1,-1] * [2,1] + cos([10,-10]) = [1,-1], \quad (3.47)
\end{aligned}$$

which is an inner approximation of $f^*([-1,1],[2,1],[10,-10]) = [0,0]$.

■

By combining inner and outer approximations, it is possible to know the quality of the results by calculating the distance between the inner approximation and the outer approximation.

3.2.1.8 Modal interval inclusion test

Definition 3.2.7. *From Theorem 3.2.1 and the notion of interpretable rational interval function stated by Equation 3.43 and 3.46, a* modal interval inclusion test $T^*(\boldsymbol{X})$ *for Equation 3.3 is defined by,*

$$T^*(\boldsymbol{X}) = \begin{cases} true & if\ OutR(fR(\boldsymbol{U}D, \boldsymbol{V}Dt)) \subseteq [0,0], \\ false & if\ InnR(fR(\boldsymbol{U}Dt, \boldsymbol{V}D)) \nsubseteq [0,0], \\ undefined & otherwise, \end{cases} \quad (3.48)$$

■

Remark 3.2.4. *The definition of* modal interval inclusion test *is an original contribution of this thesis.*

■

Example 3.2.10. *Given the logical statement*

$$(\forall u \in [0,3]')(\exists v \in [2,8]')(\exists z \in [-4,9]')f(u,v,z) = 0, \tag{3.49}$$

where $f(u,v,z)$ is the continuous function

$$f(u,v,z) = u^2 + v^2 + uv - 20u - 20 * v + 100 - 10sin(z). \tag{3.50}$$

This function can be written as

$$f(u,v,z) = u_1^2 + v_1^2 + u_2v_2 - 20u_3 - 20 * v_3 + 100 - 10sin(z), \tag{3.51}$$

*where the subindexes represent the different multi-incidences. Let us approximate the *-semantic extension $f^*([0,3],[8,2],[9,-4])$ by means of an interpretable rational interval function $fR([0,1],[8,2],[9,-4])$. First of all, the monotony of f with respect to each variable and with respect to each one of its incidences, considered as different variables, has to be computed.*

$$\begin{aligned}
&\partial f(u,v,z)/\partial u = 2u + v - 20 \in 2 * [0,3]' + [2,8]' - 20 = \\
&[-18,-6]' \leq 0, \\
&\partial f(u,v,z)/\partial u_1 = 2u \in 2 * [0,3]' = [0,6]' \geq 0, \\
&\partial f(u,v,z)/\partial u_2 = v \in [2,8]' = [2,8]' \geq 0, \\
&\partial f(u,v,z)/\partial u_3 = -20 \leq 0, \\
&\partial f(u,v,z)/\partial v = 2v + u - 20 \in 2 * [2,8]' + [0,3]' - 20 = \\
&[-16,-1]' \leq 0, \\
&\partial f(u,v,z)/\partial v_1 = 2v \in 2 * [2,8]' = [4,16]' \geq 0, \\
&\partial f(u,v,z)/\partial v_2 = u \in [0,3]' \geq 0, \\
&\partial f(u,v,z)/\partial v_3 = -20 \leq 0, \\
&\partial f(u,v,z)/\partial z = -10cos(z) \in -10 * cos([9,-4]) = [-10,10]'.
\end{aligned} \tag{3.52}$$

Notice that for computing the monotony, only the domains of the variables are taken into account and not the modality. Then, applying the corresponding D and t transformations, the next approximation is obtained

$$
\begin{aligned}
& fR(\boldsymbol{U}D, \boldsymbol{V}Dt) = \\
& Dual(U_1)^2 + Dual(V_1)^2 + Dual(U_2) * Dual(V_2) - 20 * U_3 - \\
& 20 * V_3 + 100 - 10 * sin(Z) = \\
& [3,0]^2 + [2,8]^2 + [3,0] * [2,8] - 20 * [0,3] - 20 * [8,2] + 100 - \\
& 10 * sin([9,-4]) = [290,-6],
\end{aligned} \tag{3.53}
$$

which is an interpretable calculus for f^*. In accordance with Theorem 3.2.1, as $fR([0,1],[8,2],[9,-4]) = [29,-6] \subseteq [0,0]$ is true, the logical formula is satisfied.

■

Example 3.2.11. *Consider again Example 3.2.10 but changing the interval U to $[0,6]$. In this case, f is not totally monotonic for none of its variables, therefore the duality operator cannot be applied and instead of that, the t transformation is applied on the variable V. Thus, by applying the corresponding transformations, the following calculation is obtained,*

$$
\begin{aligned}
& fR(\boldsymbol{U}D, \boldsymbol{V}Dt) = \\
& U_1^2 + Point(V_1)^2 + U_2 * Point(V_2) - 20 * U_3 - \\
& 20 * Center(V_3) + 100 - 10 * sin(z) = \\
& [0,6]^2 + [5,5]^2 + [0,6] * [5,5] - 20 * [0,6] - \\
& 20 * [5,5] + 100 - 10 * sin([9,-4]) = [-850,810].
\end{aligned} \tag{3.54}
$$

Notice that the obtained result is a very over-estimated approximation of $f^*([0,6],[8,2],[9,-4]) = [2,-6]$. In spite of being the logical formula true, it cannot be verified with the *obtained approximation.*

An inner approximation for the same example can be obtain through the following calculation

$$\begin{aligned}
&fR(\boldsymbol{U}Dt, \boldsymbol{V}D) = \\
&Point(U_1)^2 + V_1^2 + Point(U_2) * V_2 - 20 * Center(U_3) - \\
&20 * V_3 + 100 - 10 * sin(z) = \\
&[3,3]^2 + [8,2]^2 + [3,3] * [8,2] - 20 * [3,3] - \\
&20 * [8,2] + 100 - 10 * sin([9,-4]) = [107,-111].
\end{aligned} \tag{3.55}$$

As $[107,-111] \subseteq [0,0]$*, the logical formula cannot be disproved.*

■

It is important to remark that an alternative formulation to the MIA theory, and referred to as *generalized intervals*, has been proposed by Goldsztejn in Goldsztejn *et al.* (2005). This new formulation presents MIA in a slightly different way, but the main results are almost the same and the difference remains on the proofs of the theorems.

3.3 f^* algorithm

As shown in the previous section, computing the *-semantic extension (f^*) of a continuous function f by means of any interpretable rational extension can cause an overestimation of the interval evaluation, due to possible multiple occurrences of a variable. In this section, an algorithm based on MIA and branch-and-bound techniques, which allows to approximate f^* by computing inner and outer approximations, is described.

Remark 3.3.1. *The original idea of the f^* algorithm was proposed by Trepat in his PhD thesis Trepat (1982). This initial algorithm was extremely inefficient and could only solve very elemental problems. In this section, a much more efficient algorithm, which introduces many theoretical and algorithmic improvements with respect to the original algorithm, is presented. This new algorithm, together with the associated theorem, are original contributions of this thesis and a paper presenting them has been submitted for publication in a journal* Sainz et al. *(2006b).*

■

3.3.1 Key theorem

Let $\boldsymbol{X} = (\boldsymbol{U}, \boldsymbol{V})$ be a modal interval vector split into its proper ($\boldsymbol{U}$) and improper ($\boldsymbol{V}$) components. Let $\{\boldsymbol{U}_1, \dots, \boldsymbol{U}_r\}$ be a partition of $\boldsymbol{U}$ and, for every $j = 1, \dots, r$, let $\{\boldsymbol{V}_{1_j}, \dots, \boldsymbol{V}_{s_j}\}$ be a partition of $\boldsymbol{V}$. Each interval $\boldsymbol{U}_j \times \boldsymbol{V}_{k_j}$ is called a *Cell*, each $\boldsymbol{V}_{*_j}$-partition is called a *Strip*, and the $\boldsymbol{U}$-partition is called the *StripSet*.

Figure 3.1 shows a geometrical representation of an example of these partitions, when $\boldsymbol{X}$ has only one proper component and one improper component.

The algorithm we present is based on the following theorem.

Theorem 3.3.1. *Given a $\mathbb{R}^n$ to $\mathbb{R}$ real continuous function f, then*

$$\begin{aligned}&\bigvee_{j\in\{1,\dots,r\}}\ \bigwedge_{k_j\in\{1_j,\dots,s_j\}} [InnR(fR(\check{\boldsymbol{u}}_j, \boldsymbol{V}_{k_j}))] \subseteq f^*(\boldsymbol{X}) \subseteq \\ &\bigvee_{j\in\{1,\dots,r\}}\ \bigwedge_{k_j\in\{1_j,\dots,s_j\}} [OutR(fR(\boldsymbol{U}_j, \check{\boldsymbol{v}}_{k_j}))], \qquad (3.56)\end{aligned}$$

where $\check{\boldsymbol{u}}_j$ is any element of $\boldsymbol{U}'_j$ ($j = 1, \dots, r$) and $\check{\boldsymbol{v}}_{k_j}$ is any element of $\boldsymbol{V}'_{k_j}$ ($k_j = 1_j, \dots, s_j$), for example the midpoints of the intervals.

■

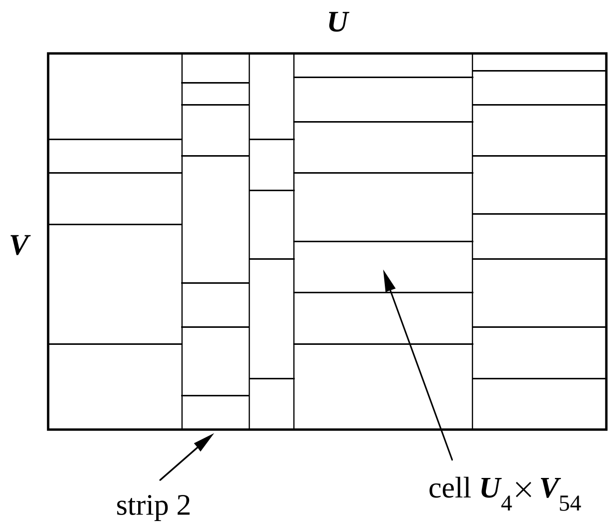


Figure 3.1: *Strip* and *Cell* partitions.

Proof. Concerning the outer approximation,

$$
\begin{aligned}
f^*(\boldsymbol{X}) &:= \bigvee_{\boldsymbol{u}\in \boldsymbol{U}'} \bigwedge_{\boldsymbol{v}\in \boldsymbol{V}'} [f(\boldsymbol{u},\boldsymbol{v})] \\
&= \bigvee_{j\in\{1,\ldots,r\}} \bigvee_{\boldsymbol{u}_j\in \boldsymbol{U}'_j} \bigwedge_{\boldsymbol{v}\in \boldsymbol{V}'} [f(\boldsymbol{u}_j,\boldsymbol{v})] && (3.57) \\
&= \bigvee_{j\in\{1,\ldots,r\}} \bigvee_{\boldsymbol{u}_j\in \boldsymbol{U}'_j} \bigwedge_{k_j\in\{1_j,\ldots,s_j\}} \bigwedge_{\boldsymbol{v}_{k_j}\in \boldsymbol{V}'_{k_j}} [f(\boldsymbol{u}_j,\boldsymbol{v}_{k_j})] && (3.58) \\
&\subseteq \bigvee_{j\in\{1,\ldots,r\}} \bigwedge_{k_j\in\{1_j,\ldots,s_j\}} [f^*(\boldsymbol{U}_j,\check{\boldsymbol{v}}_{k_j})] && (3.59) \\
&\subseteq \bigvee_{j\in\{1,\ldots,r\}} \bigwedge_{k_j\in\{1_j,\ldots,s_j\}} [OutR(fR(\boldsymbol{U}_j,\check{\boldsymbol{v}}_{k_j}))], && (3.60)
\end{aligned}
$$

where $\check{\boldsymbol{v}}_{k_j}$ is any fixed point of $\boldsymbol{V}'_{k_j}$ ($k_j = 1_j, \ldots, s_j$), for example the mid-points or the bounds of the intervals, and fR is the modal rational interval extension of the function f, because

- Equation 3.57 is true in accordance with the associativity of the join operator,

- Equation 3.58 is true in accordance with the associativity of the meet operator.

- Equation 3.59 is true since $[f(\boldsymbol{u}_j, \boldsymbol{v}_{k_j})] = f^*(\boldsymbol{u}_j, \boldsymbol{v}_{k_j}) \subseteq f^*(\boldsymbol{U}_j, \boldsymbol{v}_{k_j})$ implies

$$\begin{aligned}\bigvee_{j\in\{1,\dots,r\}} \bigvee_{\boldsymbol{u}_j\in\boldsymbol{U}'_j} \bigwedge_{k_j\in\{1_j,\dots,s_j\}} \bigwedge_{\boldsymbol{v}_{k_j}\in\boldsymbol{V}'_{k_j}} [f(\boldsymbol{u}_j, \boldsymbol{v}_{k_j})] \subseteq \\ \subseteq \bigvee_{j\in\{1,\dots,r\}} \bigwedge_{k_j\in\{1_j,\dots,s_j\}} f^*(\boldsymbol{U}_j, \check{\boldsymbol{v}}_{k_j}). \qquad (3.61)\end{aligned}$$

- Equation 3.60 is true because

$$f^*(\boldsymbol{U}_j, \check{\boldsymbol{v}}_{k_j}) \subseteq Out(fR(\boldsymbol{U}_j, \check{\boldsymbol{v}}_{k_j})). \qquad (3.62)$$

The final relation Equation 3.60 is equivalent to

$$f^*(\boldsymbol{X}) \subseteq \bigvee_{j\in\{1,\dots,r\}} \bigwedge_{k_j\in\{1_j,\dots,s_j\}} OutR(fR(\boldsymbol{U}_j, \check{\boldsymbol{v}}_{k_j})) \qquad (3.63)$$

for any partition of $\boldsymbol{X}$. Moreover, the finer partition, the better approximations.

Concerning the inner approximation

$$\begin{aligned} f^*(\boldsymbol{X}) &:= \bigvee_{\boldsymbol{u}\in\boldsymbol{U}'} \bigwedge_{\boldsymbol{v}\in\boldsymbol{V}'} [f(\boldsymbol{u}, \boldsymbol{v})] \\ &= \bigvee_{j\in\{1,\dots,r\}} \bigvee_{\boldsymbol{u}_j\in\boldsymbol{U}'_j} \bigwedge_{\boldsymbol{v}\in\boldsymbol{V}'} [f(\boldsymbol{u}_j, \boldsymbol{v})] && (3.64) \\ &= \bigvee_{j\in\{1,\dots,r\}} \bigvee_{\boldsymbol{u}_j\in\boldsymbol{U}'_j} \bigwedge_{k_j\in\{1_j,\dots,s_j\}} \bigwedge_{\boldsymbol{v}_{k_j}\in\boldsymbol{V}'_{k_j}} [f(\boldsymbol{u}_j, \boldsymbol{v}_{k_j})] && (3.65) \\ &\supseteq \bigvee_{j\in\{1,\dots,r\}} \bigwedge_{k_j\in\{1_j,\dots,s_j\}} [f^*(\check{\boldsymbol{u}}_j, \boldsymbol{V}_{k_j})]| && (3.66) \\ &\supseteq \bigvee_{j\in\{1,\dots,r\}} \bigwedge_{k_j\in\{1_j,\dots,s_j\}} [InnR(fR(\check{\boldsymbol{u}}_j, \boldsymbol{V}_{k_j}))], && (3.67) \end{aligned}$$

where $\check{\boldsymbol{u}}_j$ is any fixed point of $\boldsymbol{U}'_j$ $(j = 1, \ldots, r)$, for example the mid-points or the bounds of the intervals, and fR is the modal rational interval extension of the function f, because

- Equation 3.64 is true in accordance with the associativity of the join operator,
- Equation 3.65 is true in accordance with the associativity of the meet operator.
- Equation 3.66 is true since $[f(\boldsymbol{u}_j, \boldsymbol{v}_{k_j})] = f^*(\boldsymbol{u}_j, \boldsymbol{v}_{k_j}) \supseteq f^*(\boldsymbol{u}_j, \boldsymbol{V}_{k_j})$ implies

$$\bigvee_{j\in\{1,\ldots,r\}} \bigvee_{\boldsymbol{u}_j\in\boldsymbol{U}'_j} \bigwedge_{k_j\in\{1_j,\ldots,s_j\}} \bigwedge_{\boldsymbol{v}_{k_j}\in\boldsymbol{V}'_{k_j}} [f(\boldsymbol{u}_j, \boldsymbol{v}_{k_j})] \supseteq$$
$$\supseteq \bigvee_{j\in\{1,\ldots,r\}} \bigvee_{\boldsymbol{u}_j\in\boldsymbol{U}'_j} \bigwedge_{k_j\in\{1_j,\ldots,s_j\}} f^*(\boldsymbol{u}_j, \boldsymbol{V}_{k_j})$$
$$\supseteq \bigvee_{j\in\{1,\ldots,r\}} \bigwedge_{k_j\in\{1_j,\ldots,s_j\}} f^*(\check{\boldsymbol{u}}_j, \boldsymbol{V}_{k_j}) \qquad (3.68)$$

- Equation 3.67 is true because

$$f^*(\check{\boldsymbol{u}}_j, \boldsymbol{V}_{k_j}) \supseteq Inn(fR(\check{\boldsymbol{u}}_j, \boldsymbol{V}_{k_j})) \qquad (3.69)$$

The final Equation 3.67 is equivalent to

$$f^*(\boldsymbol{X}) \supseteq \bigvee_{j\in\{1,\ldots,r\}} \bigwedge_{k_j\in\{1_j,\ldots,s_j\}} InnR(fR(\check{\boldsymbol{u}}_j, \boldsymbol{V}_{k_j})) \qquad (3.70)$$

for any partition of $\boldsymbol{X}$. Moreover, the finer partition, the better approximations.

□

Example 3.3.1. *Table 3.1 represents a possible bisection configuration over the variables' space of Example 3.2.11, where the columns represent the* $\{U_1, \ldots, U_3\}$

partition of $U = [0, 6]$ and the rows are the corresponding $\{V_{1_j}, \ldots, V_{3_j}\}$ partitions of $V = [8, 2]$ for every U_j. Each cell contains the corresponding approximations. The last row contains the approximations which result from applying the meet operator over the cells of each columns and finally, the bottom-left cell contains the approximations resulting from applying the join operator to the obtained approximations for each column. This last result is an approximation of $f^*(U, V)$.

Table 3.1: Illustrative example of Theorem 3.3.1.

(u, v)	$u = [0, 2]$	$u = [2, 4]$	$u = [4, 6]$
$v = [8, 6]$	$Out = [-21, 17]$ $Inn = [43, -47]$	$Out = [-43, 3]$ $Inn = [27, -67]$	$Out = [-57, -3]$ $Inn = [19, -79]$
$v = [6, 4]$	$Out = [-5, 29]$ $Inn = [17, 51]$	$Out = [-31, 11]$ $Inn = [-11, 27]$	$Out = [-49, 1]$ $Inn = [-31, 11]$
$v = [4, 2]$	$Out = [19, 49]$ $Inn = [-9, 33]$	$Out = [-11, 27]$ $Inn = [-33, 13]$	$Out = [-33, 13]$ $Inn = [-49, 1]$
$\mathbf{Out} = [\mathbf{-33}, \mathbf{17}]$ $\mathbf{Inn} = [\mathbf{19}, \mathbf{-47}]$	$Out = [19, 17]$ $Inn = [43, -47]$	$Out = [-11, 3]$ $Inn = [27, -67]$	$Out = [-33, -3]$ $Inn = [19, -79]$

It can be observed that the resulting approximation is better than the one obtained in Example 3.2.11 ($Out = [-850, 810], Inn = [107, -111]$). However, it is still not good enough to decide for the truth value of the corresponding logical formula and further bisections are required.

■

3.3.2 Basic algorithm

In accordance with Theorem 3.3.1, the necessary steps for the implementation of an algorithm which computes an inner and an outer approximation of f^* is shown in Algorithm 2. In order to simplify the algorithm presentation, the following notation and concepts are introduced.

- $Inn(Cell)$: Inner approximation of a $Cell$.
- $Out(Cell)$: Outer approximation of a $Cell$.
- $Inn(Strip)$: Inner approximation of a $Strip$.
- $Out(Strip)$: Outer approximation of a $Strip$.
- $Width(\boldsymbol{U})$: Function returning the size of the widest component of a $\boldsymbol{U}$ partition.
- $Tolerance(Inner, Outer)$: Function returning the distance between the inner and the outer approximation and defined by

$$\begin{aligned}&Tolerance(Inner, Outer) := \\ &max(|Inf(Outer) - Inf(Inner)|, |Sup(Outer) - Inf(Inner)|),\end{aligned} \tag{3.71}$$

  where Inf and Sup are the respectively the left and right bounds of the corresponding approximations.

- Enqueue: The result of adding an element to a list.
- Dequeue: The result of extracting an element from a list.
- Compute inner and outer approximation of $Cell$, that is

$$Inn(Cell) := InnR(fR(\check{\boldsymbol{u}}, \boldsymbol{V})), \tag{3.72}$$

$$Out(Cell) := OutR(fR(\boldsymbol{U}, \check{\boldsymbol{v}})). \tag{3.73}$$

- Compute inner and outer approximations of $Strip$, that is

$$Inn(Strip) := \bigwedge_{\{Cell\ in\ Strip\}} Inn(Cell), \tag{3.74}$$

$$Out(Strip) := \bigwedge_{\{Cell\ in\ Strip\}} Out(Cell). \tag{3.75}$$

- Compute global inner and outer approximations, that is

$$Inner := \bigvee_{\{Strip\ in\ StripSet\}} Inn(Strip), \tag{3.76}$$

$$Outer := \bigvee_{\{Strip\ in\ StripSet\}} Out(Strip). \tag{3.77}$$

Algorithm 2 f^* algorithm

```
Input: Continuous function $f(\boldsymbol{u}, \boldsymbol{v})$ with its associated vectors of proper and
    improper intervals $(\boldsymbol{U}, \boldsymbol{V})$. Desired precision for the output $(\varphi)$.
Output: $Inner$ and $Outer$ approximations of $f^*(\boldsymbol{U}, \boldsymbol{V})$.
 1: Create a $Cell := (\boldsymbol{U}, \boldsymbol{V})$ and compute its inner and outer approximations.
    Create a $Strip$ containing $Cell$. Compute the $Strip$ approximations and
    insert $Strip$ into the $StripSet$. Compute $Inner$ and $Outer$ approximations.
 2: while $Tolerance(Inner, Outer) > \varphi$ do
 3:     Select the first $Strip$ from the $StripSet$ and the first $Cell$ from the $Strip$.
 4:     if $Width(\boldsymbol{V}) > Width(\boldsymbol{U})$ from $Cell$ then
 5:         Bisect $\boldsymbol{V}$ by the widest component, dequeue $Cell$, compute the ap-
            proximations of the resulting cells and enqueue them to the $Strip$.
            Compute inner and outer approximations of $Strip$.
 6:     else
 7:         Bisect $\boldsymbol{U}$ by the widest component and create two new strips, de-
            queue $Strip$. Compute the cells' approximations of the resulting
            strips and the approximations of the strips. Add the resulting strips
            to the $StripSet$.
 8:     end if
 9:     Compute global inner and outer approximations.
10: end while
11: return $Inner$ and $Outer$.
```

3.3.2.1 Bounding criteria

As any branch-and-bound algorithm, bounding criterions are desired in order to avoid the combinatorial blow-up. The following non-bisection criteria can be considered to avoid useless bisections:

- a $Cell$ is not bisected $Inn(Cell) \supseteq Out(Strip)$, because no division of any improper component $\boldsymbol{V}$ will improve the approximation.

Similarly,

- a $Strip$ is not bisected when $Out(Strip) \subseteq Inner$, because no division through any proper component $\boldsymbol{U}$ will improve the approximation. Moreover, this $Strip$ can be eliminated from the $StripSet$.

3.3.2.2 Stopping criteria

Apart of the tolerance criterium, the algorithm stops when the width of all the cell dimensions is smaller than a fixed precision ($Width(\boldsymbol{X}) \leq \epsilon$). Moreover, when a cell dimension reaches ϵ, this dimension is any more bisected.

When the f^* algorithm is used for proving first-order logic formulas, an extra stopping criteria can be applied, which stops the algorithm when the corresponding logical formula is satisfied or not. Notice that it is not necessary to achieve the specified $Tolerance(Inner, Outer)$ to prove the satisfaction of a logical formula. Therefore, from the notion of modal interval inclusion test in Section 3.2.1.8, the two following conditions can be introduced inside the **While** loop,

- **If** $Outer \subseteq A$ **Then Break**,
- **Else if** $Inner \nsubseteq A$ **Then Break**,

where A is the auxiliary interval (see Equation 3.48).

Test case 3.3.1. *Consider the Example 3.2.11. By using the proposed basic algorithm and after 600 seconds on a Pentium IV M, the following approximation of f^* is achieved.*

$Inner$	$[98, -67]$
$Outer$	$[-12, 18]$
$Tolerance(Inner, Outer)$	110
Number of bisections	$\simeq$ 2.000.000

Despite of the high computation time, it has not been possible to prove the satisfaction of the logical formula expressed by Equation 3.49. It seems obvious that the proposed algorithm is far from being useful. Next section introduces a set of improvements which can drastically reduce the computation effort.

■

3.3.3 Improvements

In order to make the f^* algorithm suitable for practical application, a set of strategies have been introduced. Basically, these strategies try to reduce as much as possible the number of bisections and to obtain better local approximations of the resulting partitions. Some of these improvements are simple algorithmic tricks and others are based on important results of MIA.

From now on, Example 3.2.11 will be used to quantify the improvements provoked by the different proposed strategies with respect to the basic algorithm. Each strategy will be tested incrementally with respect to the others. The comparison criteria will be the computation time and the number of bisections required for proving the logical statement involved in Example 3.2.11.

3.3.3.1 Selection strategy

Instead of using a FIFO strategy (First In, First Out) for selecting a $Strip$ and a $Cell$ from its respective containers, a more efficient strategy is proposed. It consists of selecting the $Strip$ and the $Cell$ with the biggest $Tolerance(Inn(.), Out(.))$, and which approximations match at least one of its bounds with one of the bounds of the global approximation (*Inner* or *Outer*). Using this selection strategy, a faster and more uniform convergence to the f^* value is achieved.

Test case 3.3.2. *Using the new selection strategy, the result from Table 3.2 has been obtained in 4 seconds. It can be observed that the reduction of computation*

Table 3.2: Selection strategy improvement.

$Inner$	$[4.1848, -9.4659]$
$Outer$	$[0.2163, -5.0535]$
$Tolerance(Inner, Outer)$	4.4124
Number of bisections	2109

time is drastic. Moreover, it has been proven that the logical formula stated in Example 3.2.11 is satisfied because $Outer \subseteq [0, 0]$ is true.

■

3.3.3.2 Monotonicity study

A set of additional criteria, based on the study of the monotonicity of the objective function f have been derived. By computing the partial derivatives of the objective function with regard to each variable and each of their incidences, it is possible to improve the local approximations of the $Cell$ as

$$Inn(Cell) := InnR(fR(\boldsymbol{U}Dt, \boldsymbol{V}D)) \tag{3.78}$$

and

$$Out(Cell) := OutR(fR(\boldsymbol{U}D, \boldsymbol{V}Dt)), \tag{3.79}$$

where D and t represent, respectively, the D-transformation and t-transformation (see Section 3.2.1.7).

Remark 3.3.2. *In order to compute the inner approximation $fR(\boldsymbol{U}Dt, \boldsymbol{V}D)$, the D-transformation over a variable of a $\boldsymbol{U}$ partition (Strip), requires the variable* *to be totally monotonic along the* $\boldsymbol{U}$ *partition.*

■

Moreover, it is possible to apply the following non-division criteria for a *Cell* or a *Strip*:

- If the function is totally monotonic with regard to an improper component of a *Cell*, then do not bisect the *Cell* through this improper component.
- If the function is totally monotonic with regard to a proper component along the *Strip*, then do not bisect the *Strip* through this proper component.

Test case 3.3.3. *Considering the previous criteria based on the monotony study of the function, the result from Table 3.3 has been obtained in* 0.01 *seconds. It can be observed that the reduction of computation time is significant.*

Table 3.3: Monotonicity study improvement.

$Inner$	$[29, -6]$
$Outer$	$[0.32, -4.9]$
$Tolerance(Inner, Outer)$	29
Number of bisections	8

■

3.3.3.3 Tree-optimality study

Another important improvement is based on the tree-optimality study of the rational syntactic extension fR of the continuous function f (see Section 3.2.1.7). If the vectors $\boldsymbol{U}$ and $\boldsymbol{V}$ are split into their uni-incident and multi-incident components, $(\boldsymbol{U}^{(u)}, \boldsymbol{U}^{(m)})$ and $(\boldsymbol{V}^{(u)}, \boldsymbol{V}^{(m)})$. If f is tree-optimal over a *Strip*, it is possible to introduce the improvement of computing the *Cell* approximations as

$$Inn(Cell) = InnR(fR(\boldsymbol{U}^{(u)}, \boldsymbol{U}^{(m)}Dt, \boldsymbol{V}D)), \tag{3.80}$$

$$Out(Cell) = OutR(fR(\boldsymbol{U}D, \boldsymbol{V}^{(u)}, \boldsymbol{V}^{(m)}Dt)). \tag{3.81}$$

Moreover, it is possible to apply the following non-division criterion:

- If the function is optimal with regard to a *Strip*, then do not bisect either the uni-incident improper components ($\boldsymbol{V}^{(u)}$) of its *Cells* or the uni-incident proper components ($\boldsymbol{U}^{(u)}$) of the *Strip*.

Test case 3.3.4. *Considering the previous criteria based on the optimality study of the function, the result from Table 3.4 has been obtained in less than* 0.01 *seconds. As the studied rational function is tree-optimal in all its domain, the uni-incident improper interval variable Z is not bisected anymore. Therefore, the bisection is carried out over 2 variables (u, v) instead of 3, which notably reduces the computation effort.*

Table 3.4: Tree-optimality study improvement.

$Inner$	$[29.0, -6.0]$
$Outer$	$[2.0, -6.0]$
$Tolerance(Inner, Outer)$	27
Number of bisections	2

■

Test case 3.3.5. *Consider the same example but enlarging the interval U to* $[0, 10]$*. The result from Table 3.5 is obtained in* 0.01 *seconds. In this case, it has been proven that the logical formula stated in Example 3.2.11 with $U = [0, 10]$ is not satisfied because $Inner \nsubseteq [0, 0]$ is true.*

■

Remark 3.3.3. *It can be observed that the computation time achieved by combining all the improving criterions is quite good with respect to the basic algorithm.*

Table 3.5: Result for $U = [0, 10]$.

$Inner$	$[-2.48, -45]$
$Outer$	$[-24.43, 2.25]$
$Tolerance(Inner, Outer)$	47
Number of bisections	9

However this improvement is subject to the kind of function which is considered. For instance, if the studied function is strongly non-monotonic, the monotony criterion will have a weaker effect.

■

3.3.4 Step-by-step example

Consider the logical statement from Example 3.2.11,

$$(\forall u \in [0,6]')(\exists v \in [2,8]')(\exists z \in [-4,9]')f(u,v,z) = 0, \tag{3.82}$$

where

$$f(u,v,z) = u^2 + v^2 + uv - 20u - 20v + 100 - 10sin(z), \tag{3.83}$$

is a tree-optimal function. The f^* algorithm step-by-step execution for proving that the previous logic statement is true, that is $f^*([0,6],[8,2],[9,-4]) \subseteq [0,0]$, is shown in Table 3.

Table 3 Step-by-step example of the f^* algorithm.

```
1: **Initialization**: Create $Cell_0([0,6],[8,2],[9,-4])$;
2: Insert $Cell_0$ into $Strip_0$ and $Strip_0$ into $StripSet$;
3: $Inner = Inn(Strip_0) = Inn(Cell_0) = [107,-111]$;
4: $Outer = Out(Strip_0) = Out(Cell_0) = [-85,81]$;
5: **Iteration 1**: As $Tolerance([107,-111],[-85,81]) = 192 > 0.5$;
6: Select $Strip_0$ from $StripSet$ and $Cell_0$ from $Strip_0$.
7: Bisect $Cell_0$ into $Cell_1([0,6],[8,5],[9,-4])$ and $Cell_2([0,6],[5,2],[9,-4])$;
8: $Inn(Cell_1) = [47,-81]$ and $Out(Cell_1) = [-97.75, 77.25]$;
9: $Inn(Cell_2) = [29,-21]$ and $Out(Cell_2) = [2,15]$;
10: Remove $Cell_0$ from $Strip_0$ and insert $Cell_1$ and $Cell_2$ into $Strip_0$;
11: $Inn(Strip_0) = Meet([47,-81],[29,-21]) = [47,-81]$;
12: $Out(Strip_0) = Meet([-97.75, 77.25],[2,15]) = [2,15]$;
13: $Inner = Join([47,-81]) = [47,-81]$;
14: $Outer = Join([2,15]) = [2,15]$;
15: **Iteration 2**: As $Tolerance([47,-81],[2,15]) = 96 > 0.5$, select $Strip_0$ from $StripSet$.
16: As $Tolerance(Cell_1) = 158.25 > Tolerance(Cell_2) = 36$, select $Cell_1$ from $Strip_0$.
17: As $Width([0,6]) > Width([8,5])$
18: Bisect $Strip_0$ into $Strip_1 = \{Cell_{11}([0,3],[8,5],[9,-4]), Cell_{12}([0,3],[5,2],[9,-4])\}$ and
    $Strip_2 = \{Cell_{21}([3,6],[8,5],[9,-4]), Cell_{22}([3,6],[5,2],[9,-4])\}$;
19: $Inn(Cell_{11}) = [-1,-6]$ and $Out(Cell_{11}) = [-1,-6]$;
20: $Inn(Cell_{12}) = [29,15]$ and $Out(Cell_{12}) = [29,15]$;
21: $Inn(Cell_{21}) = [40.25,-92.25]$ and $Out(Cell_{21}) = [-69.25, 17.25]$;
22: $Inn(Cell_{22}) = [13.25,-32.25]$ and $Out(Cell_{22}) = [2,-21]$;
23: $Inn(Strip_1) = Meet([-1,-6],[29,15]) = [29,-6]$;
24: $Out(Strip_1) = Meet([-1,-6],[29,15]) = [29,-6]$;
25: $Inn(Strip_2) = Meet([20.25,-72.25],[-6.75,-12.25]) = [40.25,-92.25]$;
26: $Out(Strip_2) = Meet([-89.25, 37.25],[-18,-1]) = [2,-21]$;
27: Remove $Strip_0$ from $StripSet$ and insert $Strip_1$ and $Strip_2$ into $StripSet$;
28: $Inner = Join([29,-6],[40.25,-92.25]) = [29,-6]$;
29: $Outer = Join([29,-6],[2,-21]) = [2,-6]$;
30: As $Outer = [2,-6] \subseteq [0,0]$ stop execution.
```

Figure 3.2 shows a graphical representation of the step-by-step example from Table 3.

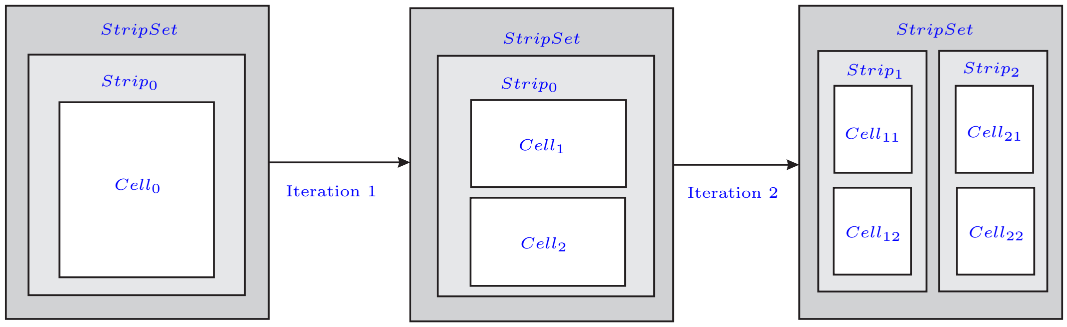


Figure 3.2: f^* algorithm step-by-step example.

3.3.5 Complexity, termination, soundness and completeness

Complexity Analysis provides a framework for understanding the cost of solving computational problems. Due to its branch-and-bound nature, the f^* algorithm presents an exponential complexity Horowitz *et al.* (1997). However, a serious study of the complexity analysis of the f^* algorithm can not be done with existing tools and therefore, it is beyond the scope of this thesis. Take for instance the famous Hansen interval algorithm for Global Optimization Hansen (1992) and notice that any complexity analysis of this algorithm is provided. The difficulty of dealing with real number for a complexity analysis is clearly illustrated by the scope of the conference CCA2006 "Computability and Complexity in Analysis" CCA2006 (2006).

Concerning the algorithm *termination*, for a given fixed precision ϵ representing the maximal bisected width dimension, the f^* algorithm finishes in less than N iteration, where $N = \prod_{i=1}^{n} \frac{Width(\boldsymbol{X}_i)}{\epsilon}$ and n is the number of variables. For a finite precision (e.g. 16 bits), the algorithm necessarily finishes because it can not produce more that N_0^n, where N_0 is the total number of representable floating point number by the machine. For example, for a 16 bits precision, $N_0 = 2^{16}$.

The f^* algorithm can be considered *sound* because it provides an inner approximation of the *-semantic extension, or what is the same, all the point of the inner approximations belongs to the solution.

The f^* algorithm is *complete* because it also provides an outer approximation of the *-semantic extension, which guaranties that all the solution points are included in the provided approximation.

3.4 Examples

The following examples try to give an idea of the different kind of problems which can be tackled with the proposed methodology for solving QRCs. Moreover, some comparisons with a state-of-the-art software Ratschan (2005) are carried out.

Example 3.4.1. *Given the QRC,*

$$\begin{aligned}&(\forall u \in [-3.1416, 3.1416]')(\exists v \in [-3.1416, 3.1416]')\\&x_1^2 sin(x_2)/exp(v) - cos(2v + u) = 0,\end{aligned} \tag{3.84}$$

where the free-variables are known to range on the interval $x_1 = [-1, 1]'$ *and* $x_2 = [-5, 5]'$*. Proving the satisfaction of this QRC is equivalent to test the truth-value of the next logical formula*

$$\begin{aligned}&(\forall x_1 \in [-1, 1]')(\forall x_2 \in [-5, 5]')(\forall u \in [-3.1416, 3.1416]')\\&(\exists v \in [-3.1416, 3.1416]')\ x_1^2 sin(x_2)/exp(v) - cos(2v + u) = 0,\end{aligned} \tag{3.85}$$

which is proven to be true in less than 2 seconds on a Pentium IV 1.5 GHz. The same QRC but with $x_1 = [-2, 2]'$*, is proven to be false in 14 seconds. It has not* *been possible to do any comparison with the available software implementations because they do not accept equality constraints.*

■

Example 3.4.2. *Given the QRC,*

$$(\forall u \in [-1,1]')(\exists v \in [-2,2]')x_1u - x_2v^2sin(x_1) > 0, \tag{3.86}$$

where the free-variables are known to range on the interval $x_1 = [0.001, 1]'$ *and* $x_2 = [-0.3, -2]'$. *Proving that this QRC is true equivalent to test the truth-value of the next logical formula*

$$\begin{aligned}&(\forall x_1 \in [0.001,1]')(\forall x_2 \in [-0.3,-2]')(\forall u \in [-1,1]')(\exists v \in [-2,2]')\\ &x_1u - x_2v^2sin(x_1) > 0, \end{aligned} \tag{3.87}$$

which is proven to be true in 0.02 seconds on a Pentium IV 1.5 GHz. The same problem is solved using RSOLVER in 2 seconds.

■

Example 3.4.3. *Given the QRC,*

$$(\exists v \in [-0.5,0.5]')[-x_1 + x_2v + x_1^2 > 0 \wedge -x_2 + (1+pow(x_1,2))v + pow(v,3) > 0], \tag{3.88}$$

where the free-variables are known to range on the intervals $x_1 = [-1, -0.5]$ *and* $x_2 = [-1, -0.5]$. *This QRC can not be solved in its original form due to the presence of shared existentially quantified variable (v) but, the equivalent QRC*

$$(\exists v \in [-0.5,0.5]')[min(-x_1 + x_2v + x_1^2, -x_2 + (1+pow(x_1,2))v + pow(v,3)) > 0], \tag{3.89}$$

can be tackled by the proposed approach. Thus, the following logical formula has to be satisfied

$$(\forall x_1 \in [-1, -0.5]')(\forall x_2 \in [-1, -0.5]')(\exists v \in [-0.5, 0.5]')$$
$$min(-x_1 + x_2 v + x_1^2, -x_2 + (1 + pow(x_1, 2))v + pow(v, 3)) > 0, \quad (3.90)$$

which is proven to be true in less than 0.01 seconds on a Pentium IV 1.5 GHz. The same problem is solved using RSOLVER with a similar computation time.

■

Remark 3.4.1. *Details on the software implementation of the f^* algorithm (FSTAR solver) and its application to the satisfaction of QRCs (QRCS solver) are explained in Chapter 8. The sources for introducing the previous examples to the corresponding solver are found in Appendix A.*

■

3.5 Conclusions

In this chapter, a new approach for proving the satisfaction of a class of Quantified Real Constraints has been presented. The way to deal with this problem is by means of the theory of Modal Interval Analysis (MIA), which allows to transform the problem of verifying a first-order logic formula into the problem of proving an interval inclusion involving modal interval computations. First, a summary of the main concepts and results of MIA are briefly introduced, with particular emphasis on the *-semantic extensions of a continuous function and the *-semantic theorem. From this two key concepts, the notion of modal interval inclusion test is introduced, which could be seen as the modal interval counterpart of the so-called interval inclusion test from the classic Interval Analysis theory. Second, an algorithm for computing outer and inner approximations of the *-semantic extension of a continuous function is presented. This algorithm is the key tool for the practical applicability of the proposed approach. Finally, some comparisons with the available state-of-the-art software implementations shows that the proposed approach can improve some aspects of the current techniques.

Chapter 4

Quantified Set Inversion Using Modal Intervals

In this chapter, a new algorithm devoted to obtain inner and outer approximations of the solution set of the class of quantified real constraints (QRCs) stated in Section 2.2.2 is presented. This algorithm, based on Modal Interval Analysis and branch-and-bound techniques, is referred to as the Quantified Set Inversion (QSI) algorithm because, it is inspired by the well-known Set Inversion Via Interval Analysis (SIVIA) algorithm.

4.1 Introduction

As already mentioned in Section 2.3.2, approximate methods can approximate the solution set Σ of a QRC by means of an interval box containing (or contained in) Σ or by means of a set of non-overlapping interval boxes, which can provide a better approximation of Σ but, suffers from a higher computational complexity. This section provides an algorithm based on the second approach.

4.1.1 Set Inversion Via Interval Analysis

The Set Inversion Via Interval Analysis algorithm (SIVIA) Jaulin & Walter (1993) is an interval branch-and-bound algorithm devoted to approximate, in its original version, solution sets of the form

$$\Sigma = \{\boldsymbol{x} \in \boldsymbol{X}' \mid \mathbf{f}(\boldsymbol{x}) \leq 0\}, \tag{4.1}$$

where f is a continuous real functions from $\mathbb{R}^n$ to $\mathbb{R}^m$. The SIVIA algorithm combines a branch-and-bound algorithm, like the one presented in Section 2.3.2.2, with the following two rules to determine if a box $\boldsymbol{X}'$ is contained in the solution set Σ or if $\boldsymbol{X}'$ does not intersect with Σ.

$$\begin{array}{ll} \textit{Inside}: & (\forall \boldsymbol{x} \in \boldsymbol{X}')\ f(\boldsymbol{x}) \leq 0 \Leftrightarrow \\ & \boldsymbol{X}' \subseteq \Sigma. \end{array} \tag{4.2}$$

$$\begin{array}{ll} \textit{Outside}: & (\forall \boldsymbol{x} \in \boldsymbol{X}')\ f(\boldsymbol{x}) > 0 \Leftrightarrow \\ & \boldsymbol{X}' \cap \Sigma = \emptyset. \end{array} \tag{4.3}$$

To proves the previous rules, the SIVIA algorithm uses the interval inclusion test defined in Section 3.2.1.8. Notice that the basic SIVIA algorithm is not able to deal with real constraints involving quantified variables like the one stated in Section 2.2.2.

Next section presents an algorithm inspired on the SIVIA algorithm, which uses the notion of modal interval inclusion test introduced in Section 3.2.1.8. This algorithm is referred to as the Quantified Set Inversion (QSI) algorithm.

Remark 4.1.1. *The QSI algorithm is an original contribution of the author of this thesis and together with some applications, it has been presented in different international conferences Calm* et al. *(2006); Flórez* et al. *(2005); Herrero* et al. *(2004a,b, 2005a) and in a journal Herrero* et al. *(2005b).*

■

4.2 Quantified Set Inversion algorithm

The QSI algorithm is devoted to approximating the solution set of the class of QRCs stated in Section 2.2.2, which most general definition is expressed by

$$\Sigma = \{\boldsymbol{x} \in \boldsymbol{X}' \mid (\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{U}') f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = 0\}, \tag{4.4}$$

where f is real function from $\mathbb{R}^n$ to $\mathbb{R}$.

Remark 4.2.1. *It is easy to see that any instance of the problem defined in Section 2.2.2 can be reduced to Equation 4.4 (see Section 3.1).*

■

The QSI algorithm is based on a branch-and-bound algorithm over the free-variables vector (see Section 2.3.2.2), and the two following bounding rules, which are used to determine if a resulting box from the bisection procedure is included in the solution set $\boldsymbol{X}' \subseteq \Sigma$ ($InsideQSI$ rule), or if $\boldsymbol{X}'$ does not intersect with the solution set $\boldsymbol{X}' \cap \Sigma = \emptyset$ ($OutsideQSI$ rule).

The first bounding rule is

$$InsideQSI: \quad \begin{array}{l}(\forall \boldsymbol{x} \in \boldsymbol{X}')(\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{V}')\ f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = 0 \Leftrightarrow \\ \boldsymbol{X}' \subseteq \Sigma.\end{array} \tag{4.5}$$

where $\boldsymbol{X}'$ is any box resulting from the bisection procedure.

Notice that the $InsideQSI$ rule is a first-order logic formula that can be tested by means of the modal interval inclusion test introduced in Section 3.2. Thus,

$$\begin{aligned} Outer(f^*(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{V})) \subseteq [0,0] &\Rightarrow f^*(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{V}) \subseteq [0,0] \\ &\Leftrightarrow (\forall \boldsymbol{x} \in \boldsymbol{X}')(\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{V}')\ f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = 0 \\ &\Leftrightarrow \boldsymbol{X}' \subseteq \Sigma, \end{aligned} \tag{4.6}$$

where $\boldsymbol{X}, \boldsymbol{U}$ are proper intervals, $\boldsymbol{V}$ in an improper interval, $Outer(f^*(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{V}))$ is an outer approximation of the *-semantic extension of the continuous function f.

The second bounding rule is

$$\begin{aligned} OutsideQSI: \quad & \boldsymbol{X}' \cap \Sigma = \emptyset \Leftrightarrow \\ & (\forall \boldsymbol{x} \in \boldsymbol{X}')\neg((\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{V}')\ f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = 0)) \Leftrightarrow \\ & (\forall \boldsymbol{x} \in \boldsymbol{X}')(\exists \boldsymbol{u} \in \boldsymbol{U}')(\forall \boldsymbol{v} \in \boldsymbol{V}')\ f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) \neq 0). \end{aligned} \tag{4.7}$$

Again, the $OutsideQSI$ rule is a first-order logic formula which can be proved by means of the modal interval inclusion test through the following sequence of implications.

$$\begin{aligned}
Inner(f^*(\boldsymbol{X},\boldsymbol{U},\boldsymbol{V})) \nsubseteq [0,0] &\Rightarrow f^*(\boldsymbol{X},\boldsymbol{U},\boldsymbol{V}) \nsubseteq [0,0] \\
&\Leftrightarrow \neg((\forall \boldsymbol{u} \in \boldsymbol{U}')(\exists \boldsymbol{v} \in \boldsymbol{V}')(\exists \boldsymbol{x} \in \boldsymbol{X}')\ f(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}) = 0) \\
&\Leftrightarrow (\exists \boldsymbol{u} \in \boldsymbol{U}')(\forall \boldsymbol{v} \in \boldsymbol{V}')(\forall \boldsymbol{x} \in \boldsymbol{X}') f(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}) \neq 0 \\
&\Rightarrow (\forall \boldsymbol{x} \in \boldsymbol{X}')(\exists \boldsymbol{u} \in \boldsymbol{U}')(\forall \boldsymbol{v} \in \boldsymbol{V}') f(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}) \neq 0 \\
&\Leftrightarrow \boldsymbol{X}' \cap \Sigma = \emptyset, \qquad (4.8)
\end{aligned}$$

with $\boldsymbol{U}$ a proper interval, $\boldsymbol{X}, \boldsymbol{V}$ improper intervals, $Inner(f^*(\boldsymbol{X},\boldsymbol{U},\boldsymbol{V}))$ an inner approximation of the the *-semantic extension of the continuous function f.

Finally, if any of the bounding rules is accomplished, the box $\boldsymbol{X}'$ is considered as undefined and is bisected. Algorithm 4 shows the QSI algorithm in pseudocode form.

Algorithm 4 QSI algorithm

Input: ϕ and ϵ.

Output: Σ_{Inn} and Σ_{Out} of the solution set of ϕ.

```
1: List:={$\boldsymbol{X}'$}; $\Sigma_{Inn} := \{\emptyset\}$; $\triangle\Sigma := \{\emptyset\}$;
2: while $List$ not empty do
3:     Dequeue $\boldsymbol{X}'$ from $List$;
4:     if $InsideQSI$ is true for $\boldsymbol{X}'$ then
5:         Enqueue $\boldsymbol{X}'$ to $\Sigma_{Inn}$;
6:     else if $OutsideQSI$ is true for $\boldsymbol{X}'$ then
7:         Do nothing;
8:     else if $Width(\boldsymbol{X}') < \epsilon$ then
9:         Enqueue $\boldsymbol{X}'$ to $\triangle\Sigma$;
10:    else
11:        Bisect $\boldsymbol{X}'$ and enqueue the resulting boxes to $List$;
12:    end if
13: end while
14: Enqueue $\Sigma_{Inn}$ and $\triangle\Sigma$ to $\Sigma_{Out}$;
```

where

- *List*: List of boxes.
- Σ_{Inn}: List of boxes such that $\Sigma_{Inn} \subseteq \Sigma$.
- Σ_{Out}: List of boxes such that $\Sigma \subseteq \Sigma_{Out}$.
- Enqueue: The result of adding a box to a list.
- Dequeue: The result of extracting a box from a list.
- $Width(\boldsymbol{X}')$: Function returning the widest relative width of $\boldsymbol{X}'$ with respect to the original box,
- ϵ: A real value representing the desired precision.

Example 4.2.1. *Given the QRC*

$$(\forall u \in [-1,1]')(\exists v \in [-2,2]')[x_1 u - x_2 v^2 sin(x_1) > 0], \tag{4.9}$$

where (x_1, x_2) *are free-variables ranging over the domains* $([-10,10]', [-10,10]')$*. Figure 4.1 shows a graphical representation of the solution provided by the QSI algorithm for an* $\epsilon = 0.05$ *in 40 seconds on a Pentium IV M 1.5 GHz, where, red boxes are included in the solution set, blue boxes are outside of the solution set and green boxes are undefined.*

■

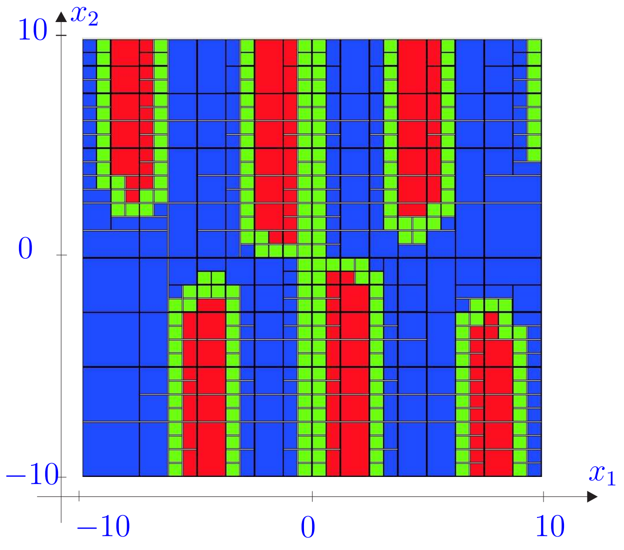


Figure 4.1: Graphical output for Example 4.2.1.

Example 4.2.2. *Given the QRC,*

$$(\exists v \in [-0.5,0.5]')[-x_1 + x_2 v + x_1^2 > 0 \wedge -x_2 + (1 + pow(x_1,2))v + pow(v,3) > 0], \tag{4.10}$$

which free-variables are known to range on the box $(x_1, x_2) = ([-10,10]', [-10,10]')$*. This QRC can not be solved by the QSI algorithm in its original form due to the* *presence of a shared existentially quantified variable (v) but, the equivalent QRC*

$$(\exists v \in [-0.5, 0.5]')$$

$$[min(-x_1 + x_2 v + x_1^2, -x_2 + (1 + pow(x_1, 2))v + pow(v, 3)) > 0], \quad (4.11)$$

can be tackled. Figure 4.2 shows a graphical representation of the solution provided by the QSI algorithm for an $\epsilon = 0.05$ *in 49 seconds on a Pentium IV M 1.5 GHz.*

■

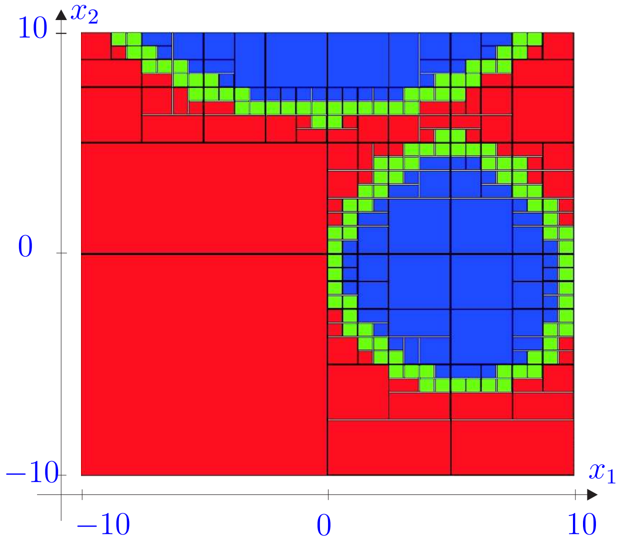


Figure 4.2: Graphical output for Example 4.2.2.

Remark 4.2.2. *The same complexity, termination, soundness and completeness properties of the SIVIA algorithm Jaulin & Walter (1993), can be applied to the QSI algorithm. Thus, the algorithm presents an exponential complexity because of its branch-and-bound nature, it guaranties termination for non ill-posed problems and a finite precision, is sound because, it provides a continuous guaranteed inner approximation the solution set and is complete because, it provides an outer* *approximation of the solution set.*

■

4.3 Application examples

This section is devoted to presenting some application examples of the QSI algorithm. In order to quantify the performance of the proposed algorithm, comparisons with some state-of-the-art techniques are carried out. Two criterions have been taken into account for the comparisons: the computation time and the volume of the resulting undefined area. However, it is very difficult to carry out a reliable comparison and only an approximative information can be provided. For instance, the computation time can be significantly different depending on the computer performance and also on the formal expression of the involved predicates. Concerning the undefined area criterion, most of the times the comparison can only be done visually through the graphical output of the solvers.

4.3.1 Robust Control

Example 1: Consider the example from Dorato (2000). In particular, consider the plant with the transfer function

$$G(s,p) = \frac{s^2 + s + (3+3p)}{s^3 + (1+p)s^2 + (1+p)s + (0.5 + p^2 + 3p)} \tag{4.12}$$

and the compensator by

$$C(s,q) = q, \tag{4.13}$$

where p is an uncertain plant parameter, $p \in [0,1]'$, q is the design parameter. The problem is to determine the admissible design parameter set Σ that guarantees robust closed-loop stability. From the Liénard-Chipart stability test one can show analytically that there is a circular instability region in the p-q space given by

$$(p-0.5)^2 + (q-0.5) \leq \rho^2, \tag{4.14}$$

where $\rho = 1/4$.

The closed-loop characteristic polynomial for this system is given by

$$D(s,p,q) = s^3 + (1+p+q)s^2 + (1+p+q)s + (0.5+\rho^2+3p+3q+2pq). \quad (4.15)$$

The Liénard-Chipart conditions applied to this polynomial require, with $\rho = 1/4$ and fractions cleared,

$$\begin{aligned} v_1(p,q) &= 9 + 48p + 48 * q + 32 * p * q \geq 0, \\ v_2(p,q) &= 1 + p + q \geq 0, \\ v_3(p,q) &= -16p - 16q + 16p^2 + 16q^2 + 7 \geq 0. \end{aligned} \quad (4.16)$$

The solution set Σ for robust closed-loop stability is then given by

$$\Sigma = \{q \in [-3,3] | (\forall p \in [0,1])[v_1(p,q) \geq 0 \wedge v_2(p,q) \geq 0 \wedge v_3(p,q) \geq 0\}. \quad (4.17)$$

The result obtained with the QSI algorithm for an $\epsilon = 0.02$ in 0.4 seconds on a Pentium IV M 1.5GHz is shown in Figure 4.3.

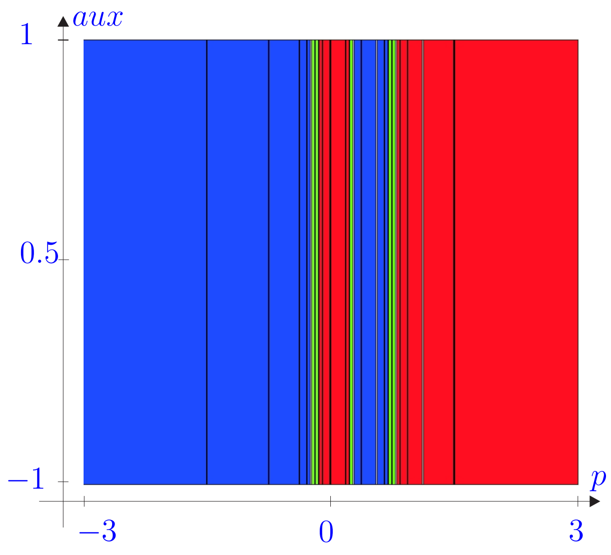


Figure 4.3: Approximation of Σ for robust closed-loop stability.

Comparisons: The same problem is resolved by RSOLVER Ratschan (2005) in 3 seconds and in 9 seconds by Proj2D solver Dao (2005) on a Pentium IV M 1.5GHz. In Figure 4.4, the graphical outputs of both solvers are shown.

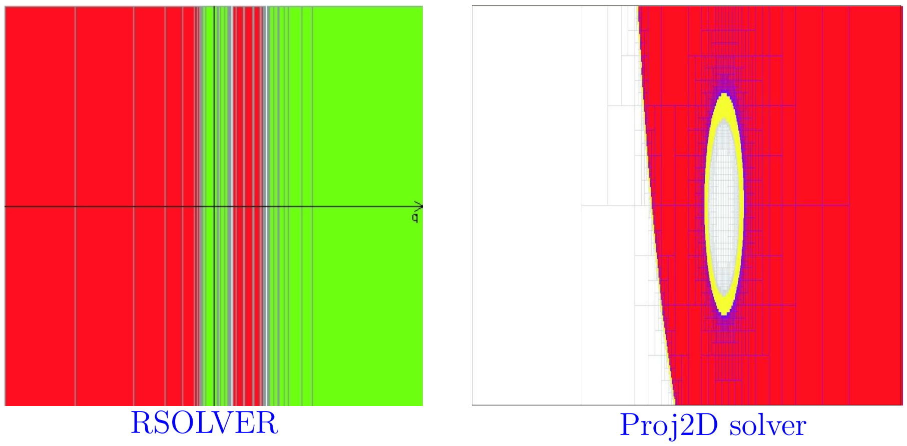


Figure 4.4: RSOLVER and Proj2D comparisons for robust closed-loop stability.

Remark 4.3.1. *Notice that the obtained figure with the Proj2D solver is different. The reason is that the Proj2D solver can not deal with universal quantified parameters and it has to be dealt as a free-variable. Nevertheless, the solution for the design parameter q (horizontal axis) is equivalent.*

■

Example 2: Consider the example from Jaulin *et al.* (2002) consisting of characterizing the set of all parameter vectors **c** of a parametric linear PI controller $\Gamma(\mathbf{c})$ that robustly stabilize an uncertain linear time-invariant model of the process represented in Figure 4.5. This problem is equivalent to characterizing the following set

$$\Sigma_c = \{\mathbf{c} \in \mathbf{C}' \mid (\forall \mathbf{p} \in \mathbf{P}') f(\mathbf{c}, \mathbf{p}) < 0\}, \tag{4.18}$$

where $f(\mathbf{c}, \mathbf{p})$ is obtained by the Routh criterion,

$$f(\mathbf{c}, \mathbf{p}) \triangleq \min \begin{pmatrix} p_2 + y_1 \\ p_2 p_3 + 1 + y_2 \\ p_2 p_3^2 + p_3 - \frac{p_2\left(p_3^2 + c_2 p_1 p_3^2\right)}{p_2 p_3 + 1} + y_3 \\ p_3^2 + c_2 p_1 p_3^2 - \frac{(p_2 p_3 + 1)^2 \left(c_1 p_1 p_3^2\right)}{\left(p_2 p_3^2 + p_3\right)(p_2 p_3 + 1) - p_2\left(p_3^2 + c_2 p_1 p_3^2\right)} + y_4 \\ c_1 p_1 p_3^2 + y_5 \end{pmatrix} \tag{4.19}$$

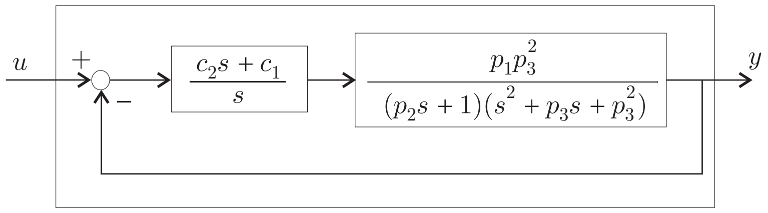


Figure 4.5: Uncertain linear time-invariant model

For $\mathbf{C}' = [0, 1]'^2$ and $\mathbf{P}' = [0.9, 1.1]'^3$ and $\epsilon = 0.02$, the result shown in Figure 4.6 is obtained by QSI algorithm in 45 seconds on a Pentium IV M 1.5GHz.

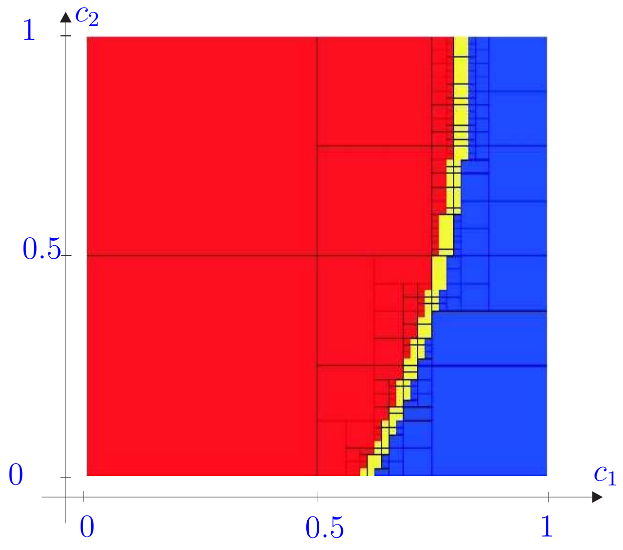


Figure 4.6: Approximation of Σ_c obtained with QSI algorithm.

Comparisons: Significant differences can be observed with the tested solvers (Projection algorithm Jaulin *et al.* (2002), RSOLVER Ratschan (2005), AQCS Ratschan (2002a)). Table 4.1 shows the computation times in seconds for each solver and for the same $\epsilon = 0.02$, together with the type of computer which have been used to execute the test. In Figure 4.7, the graphical comparisons can be observed. Notice that the solution provided by the QSI algorithm is clearly better than the one provided by the other techniques.

Table 4.1: Computation time comparisons

Solver	**Time**	**Computer**
Projection Algorithm	470	Pentium III
RSOLVER	> 300	Web Server (?)
AQCS	> 100	Pentium IV 2GHz
QSI	45	Pentium IV M 1.5GHz

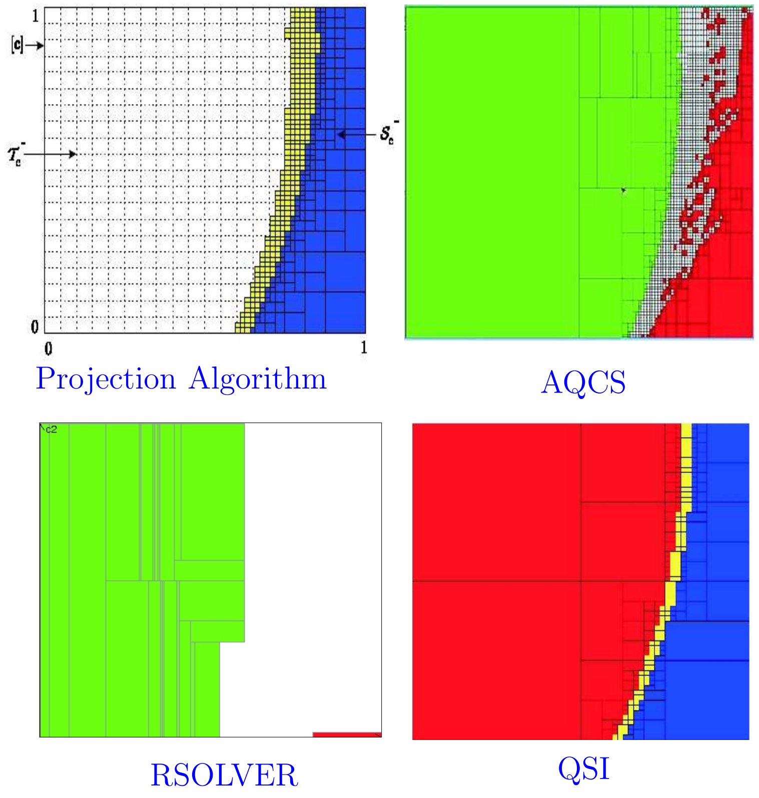


Figure 4.7: Comparisons with Projection Algorithm, RSOLVER, AQCS solver.

4.3.2 Set projection

The QSI algorithm can be used to characterize the projection of a set defined by nonlinear real predicates satisfying certain conditions.

Let Σ be the subset of $\mathbb{R}^n$ defined by:

$$\Sigma = \{(\boldsymbol{x}, \boldsymbol{v}) \in (\boldsymbol{X}', \boldsymbol{V}') | \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{v}) = 0\}. \tag{4.20}$$

The projection set of Σ over $\boldsymbol{x}$ is defined by

$$\Sigma_{\boldsymbol{x}} = \{\boldsymbol{x} \in \boldsymbol{X}' | (\exists \boldsymbol{v} \in \boldsymbol{V}') \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{v}) = 0\}. \tag{4.21}$$

Figure 4.8 is a graphical illustration of a generic projection of Σ on $\boldsymbol{x}$.

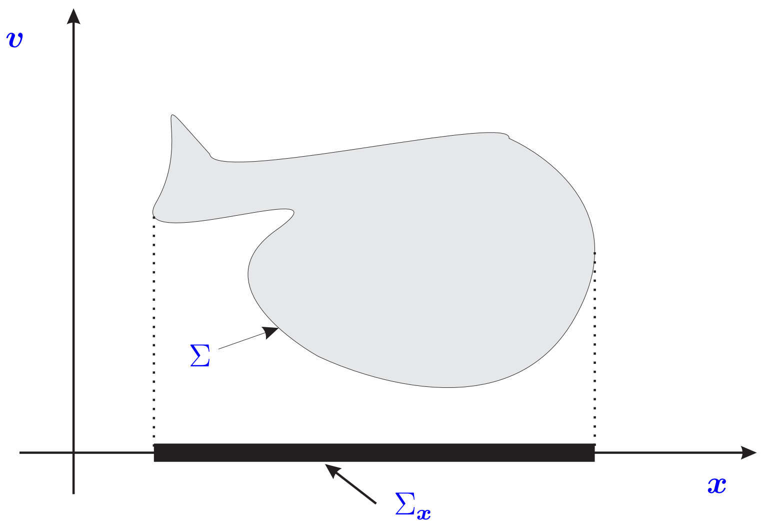


Figure 4.8: Projection of Σ on the $\boldsymbol{x}$-plane.

For example, consider the set representing a paraboloid

$$\Sigma = \{(x_1, x_2, x_3) \in [-2, 2]'^3 |\ x_1^2 - x_2^2 + x_3 = 0\}. \tag{4.22}$$

The projection set on the x_1x_2-plane is defined by

$$\Sigma_{x_1x_2} = \{(x_1, x_2) \in [-2, 2]'^2 |\ (\exists \boldsymbol{x}_3 \in [-1, 1]') x_1^2 - x_2^2 + x_3 = 0\}. \tag{4.23}$$

Figure 4.9 shows the approximated projection on the x_1x_2-plane obtained by the QSI algorithm in 2.5 seconds on a Pentium 1.5 M GHz for an $\epsilon = 0.01$. Where,

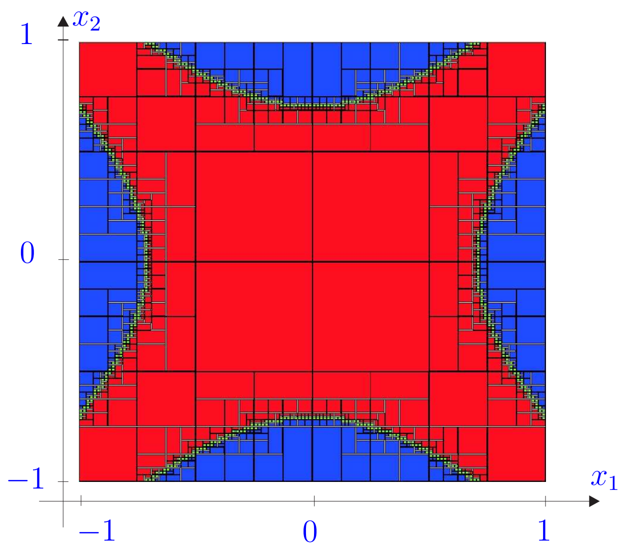


Figure 4.9: Projection of a paraboloid on the x_1x_2-plane.

the red area corresponds to an inner approximation of the paraboloid projection, the blue area is outside of the projection and the green area is undefined.

No comparisons have been carried out because the existing solvers do not support equality predicates.

4.3.2.1 Application to computer graphics

From the notion of set projection, the QSI algorithm, together with other interval based techniques like, geometric transformations (e.g. rotations, translation) and ray tracing for giving realistic appearance, is currently being applied in the domain of Computer Graphics for the visualization of 3D surfaces given by implicit functions. This work is currently being developed by Flórez in his PhD thesis Flórez *et al.* (2005).

Example 4.3.1. *Consider the 3 surfaces given by the following implicit functions.*

- ***Crosscap***: $(4x^2(x^2 + y^2 + z^2 + z) + y^2(y^2 + z^2 - 1) = 0,$
- ***Hyperbolic paraboloid***: $x^2 - y^2 + z = 0,$

- ***Quartic***: $(4(x^4 + (y^2 + z^2)^2) + 17x^2(y^2 + z^2) - 20(x^2 + y^2 + z^2) + 17) = 0$,

where the variables x, y and z range in the domain $[-1, 1]'^3$.

By applying the QSI based algorithm for a precision of $\epsilon = 1^{-3}$, the projections of Figure 4.10 (see Flórez et al. (2005)) are obtained with a computation time of 33, 37 and 213 seconds respectively.

■

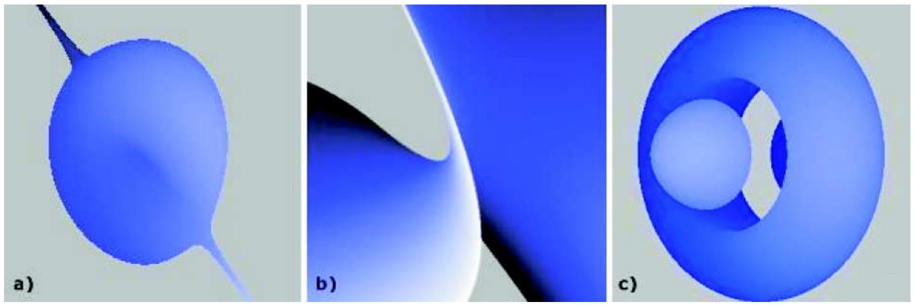


Figure 4.10: Projection of a) crosscap, b) hiperbolic paraboloid and c) quartic surfaces.

It is important to remark that the obtained results are in general better, in terms of the quality of the image (e.g. less aliasing effect) and in terms of the computation time, with respect to other classical algorithms for implicit surface visualization.

4.3.3 Bounded-error Parameter Identification

Bounded-error Parameter Identification consists of finding approximations of the solution set for the parameters of a given model that are consistent with the input/output observations. The parameter identification problem treated in this section is a well known problem of the literature Jaulin & Walter (1999), which can be defined by two main characteristics:

i. **The process model**: A nonlinear process which depends on the variable t and two parameters x_1 and x_2 is used. The theoretical model of the process is,

$$f(\boldsymbol{x}, t) = 20exp(-x_1 t) - 8exp(-x_2 t). \tag{4.24}$$

ii. **The restrictions to be satisfied**: The restrictions imposed by the system are,

$$(f(\boldsymbol{x}, t_i) \in Y_i'),\ (t_i \in T_i'),\ (\forall i \in 1, \dots, 10), \tag{4.25}$$

where Y_i' corresponds to the uncertainty associated to the measure y_i and T_i' represents the uncertainty associated to the measurement time t_i.

Table 4.2 shows the uncertainty associated to y and t represented by intervals.

The consistent parameter set is defined by

$$\begin{aligned}\Sigma = \{\boldsymbol{x} \in \boldsymbol{X}' \mid (\exists t_1 \in T_1')(\exists y_1 \in Y_1') f(\boldsymbol{x}, t_1) - y_1 = 0 \\ , \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots , \\ (\exists t_{10} \in T_{10}')(\exists y_{10} \in Y_{10}') f(\boldsymbol{x}, t_{10}) - y_{10} = 0\}.\end{aligned} \tag{4.26}$$

Grouping the existential quantifiers and expressing it under a vectorial form

$$\Sigma = \{\boldsymbol{x} \in \boldsymbol{X}' | (\exists \mathbf{t} \in \mathbf{T}')(\exists \mathbf{y} \in \mathbf{Y}') \mathbf{f}(\boldsymbol{x}, \mathbf{t}) - \mathbf{y} = 0\}, \tag{4.27}$$

where $\mathbf{f}(\boldsymbol{x}, \mathbf{t}) = (f(\boldsymbol{x}, \mathbf{t_1}), \dots, \mathbf{f}(\boldsymbol{x}, \mathbf{t_{10}}))$, $\mathbf{t} = (t_1, \dots, t_{10})$ and $\mathbf{y} = (y_1, \dots, y_{10})$.

For one sample i ($i = \{1, \dots, 10\}$), the corresponding $InsideQSI$ rule is

$$(\forall x_1 \in X_1')(\forall x_2 \in X_2')(\exists t_i \in T_i')(\exists y_i \in Y_i') f(\boldsymbol{x}, t_i) - y_i = 0, \tag{4.28}$$

which can be proven by the following inclusion test

$$Outer(f_i^*(X_1, X_2, T_i, Y_i)) \subseteq [0, 0], \tag{4.29}$$

with X_1 and X_2 proper intervals and T_i and Y_i improper intervals. And the corresponding $OutsideQSI$ rule is

$$(\forall x_1 \in X_1')(\forall x_2 \in X_2')(\forall t_i \in T_i')(\forall y_i \in Y_i') f(\boldsymbol{x}, t_i) - y_i \neq 0, \tag{4.30}$$

Table 4.2: Uncertainty associated to t and y.

i	$\mathbf{T_i}$	$\mathbf{Y_i}$
1	[-0.25,1.75]'	[2.7,12.1]'
2	[.5,2.5]'	[1.04,7.14]'
3	[1.25,3.25]'	[-0.13,3.61]'
4	[2,4]'	[-0.95,1.15]'
5	[5,7]'	[-4.85,-0.29]'
6	[8,10]'	[-5.06,-0.36]'
7	[12,14]'	[-4.1,-0.04]'
8	[16,18]'	[-3.16,0.3]'
9	[20,22]'	[-2.5,0.51]'
10	[24,26]'	[-2,0.67]'

which is implied by the following exclusion test

$$Inner(f_i^*(X_1, X_2, T_i, Y_i)) \nsubseteq [0, 0], \tag{4.31}$$

with X_1, X_2, T_i and Y_i improper intervals. Then, Σ is obtained by intersecting the resulting Σ_i for each sample time.

$$\Sigma = \Sigma_i \cap \cdots \cap \Sigma_{10}. \tag{4.32}$$

For $\boldsymbol{X} = [0, 1.2]' \times [0, 0.5]'$ and a precision of $\epsilon = 0.01$, the QSI algorithm generates, in 40 seconds on a Pentium IV M 1.5GHz, the paving of Figure 4.11.

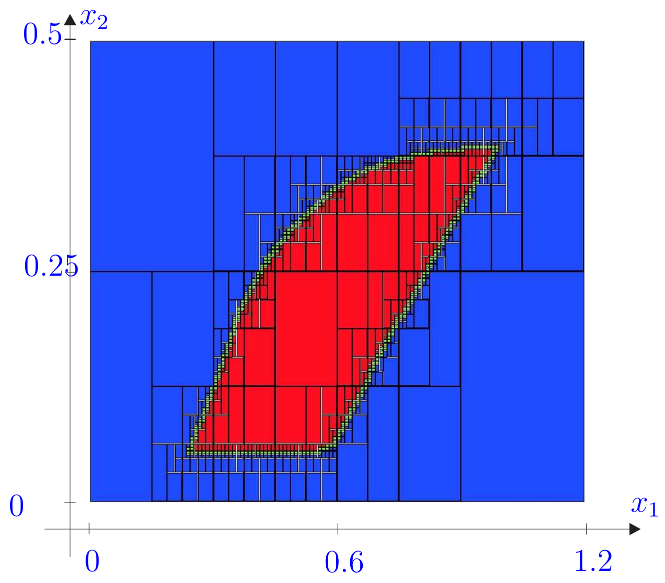


Figure 4.11: QSI output for the bounded-error parameter identification problem.

4.3.3.1 Comparisons

Comparisons with the algorithms Extended-SIVIA Jaulin & Walter (1999) and Proj2D Dao (2005), show no significant differences in terms of computation time and undefined area with respect to the QSI algorithm. However, with respect to RSOLVER Ratschan (2005), the differences are important, and after 5 minutes of computation, the resulting undefined area is much bigger than the one obtained with the other solvers. Table 4.3 shows the computation times in seconds for each solver together with the computer which have been used to carry out the test. In Figure 4.12, the solver outputs can be graphically observed.

Table 4.3: Computation time comparisons.

Solver	**Time**	**Computer**
Extended-SIVIA	38	Pentium I 133MHz
Proj2D	30	Pentium IV M 1.5GHz
RSOLVER	> 300	Web Server (?)
QSI	40	Pentium IV M 1.5GHz

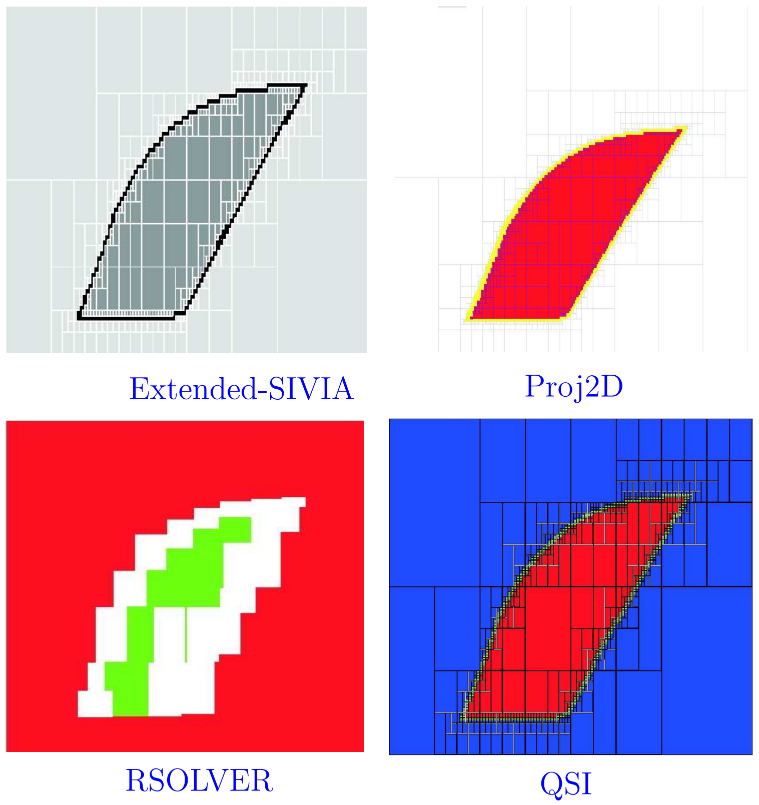


Figure 4.12: Extended-SIVIA, Proj2D and RSOLVER comparisons.

Remark 4.3.2. *The Extended-SIVIA, RSOLVER and Proj2D solvers, cannot directly solve the problem from Equation 4.27 because of the presence of equality predicates. However, in this special case, it is possible to use the following transformation in order to convert the equality predicates into inequality predicates.*

$$\Sigma = \{\boldsymbol{x} \in \boldsymbol{X}' | (\exists \mathbf{t} \in \mathbf{T}')(\exists \mathbf{y} \in \mathbf{Y}')\mathbf{y}(\boldsymbol{x}, \mathbf{t}) - \mathbf{y} = 0\} \Leftrightarrow$$
$$\{\boldsymbol{x} \in \boldsymbol{X}' | (\exists \mathbf{t} \in \mathbf{T}')(\exists \mathbf{y} \in \mathbf{Y}')\mathbf{y}(\boldsymbol{x}, \mathbf{t}) = \mathbf{y}\} \Leftrightarrow$$
$$\{\boldsymbol{x} \in \boldsymbol{X}' | (\exists \mathbf{t} \in \mathbf{T}')[\mathbf{y}(\boldsymbol{x}, \mathbf{t}) \geq Inf(\mathbf{Y}') \wedge \mathbf{y}(\boldsymbol{x}, \mathbf{t}) \leq Sup(\mathbf{Y}')]\}. \quad (4.33)$$

■

4.3.4 Aircraft control

Consider the motivation example from Section 1.1 referring to the equilibrium of an aircraft. The aerodynamic moments acting over the aircraft T_L, T_M and T_N are usually given in tabular form together with some interpolation method. In Stevens & Lewis (1993) these tables are listed for an F-16 aircraft and the following are scaled polynomial approximations of the corresponding functions

$$\begin{aligned}
T_L(\alpha, \beta, u_1, u_3) = \quad & -q_1\beta - q_2\alpha\beta + q_3a^2\beta + q_4\beta^3 + u_1(-q_5 - q_6\alpha + \\
& q_7\alpha^2 - q_8\alpha^3 + q_9\beta^2 + q_{10}\alpha\beta^2) + u_3(q_{11} - q_{12}\alpha + \\
& q_{13}\alpha^2 - q_{14}\alpha^3 + q_{15}\alpha^4 - q_{16}\beta^2 - q_{17}\alpha\beta^2 + \\
& q_{18}\alpha^2\beta^2 + q_{19}\beta^4), \quad (4.34) \\
T_M(\alpha, u_2) = \quad & -q_{20} - q_{21}u_2 + u_2^2 + q_{22}u_2^3 + q_{23}\alpha - q_{24}u_2\alpha + \\
& q_{25}u_2^2\alpha - q_{26}\alpha^2 + q_{27}\alpha^2 + q_{28}\alpha^3, \quad (4.35) \\
T_N(\alpha, \beta, u_1, u_3) = \quad & q_{29}\beta - q_{30}\alpha\beta - q_{31}\alpha^2\beta + q_{32}\alpha^3\beta - q_{33}\beta^3 + \\
& q_{34}\alpha\beta^3 + u_1(-q_{35} + q_{36}\alpha - q_{37}\alpha^2 + q_{38}\beta^2 + \\
& q_{39}\alpha^3 - q_{40}\alpha\beta^2) + u_3(-q_{41} + q\alpha - \\
& q_{42}\alpha^2 + q_{43}\beta^2 + q_{44}\alpha^3 + q_{45}\alpha\beta^2 - q_{46}\alpha^4 - \\
& q_{47}\alpha^2\beta^2 - q_{48}\beta^4), \quad (4.36)
\end{aligned}$$

where $\{q_1, \cdots, q_{48}\}$ are polynomial coefficients, which in the present work are considered uncertain in contrast with the original work.

Its solution is expressed by

$$\begin{aligned}
&\Sigma = \{(\alpha,\beta)|(\forall q_1 \in Q'_1)\cdots(\forall q_{48} \in Q'_{48})(\exists u_1 \in U'_1)(\exists u_2 \in U'_2)(\exists u_3 \in U'_3)\\
&(T_L(\alpha,\beta,u_1,u_3,q_1,\cdots,q_{19}) = 0 \wedge T_M(\alpha,u_2,q_{20},\cdots,q_{28}) = 0 \wedge\\
&T_N(\alpha,\beta,u_1,u_3,q_{29},\cdots,q_{48}) = 0).\}
\end{aligned} \tag{4.37}$$

Consider the problem stated above of finding the admissible set of orientation (α,β) of a F-16 aircraft for which the control-surface system (u_1,u_2,u_3) can keep the aircraft stabilized. Let us suppose an initial search domain $(\alpha,\beta) \in ([0,1]',[-1,1]')$ and the uncertain coefficients of Table 4.4.

Notice that the equality predicates T_L and T_N from Equation 4.37 share the existential variables u_1 and u_3. However, doing some simple algebraic substitutions, which consist of isolating u_3 in T_L and replacing it in T_N, it is possible to transform the two predicates into a single predicate, which yields the equivalent expression

$$\begin{aligned}
&\Sigma = \{\alpha\times\beta|(\forall q_1 \in Q'_1)\cdots(\forall q_{48} \in Q'_{48})(\exists u_2 \in U'_2)(\exists u_3 \in U'_3)\\
&(T_S(\alpha,\beta,u_3,q_1,\cdots,q_{19}) = 0 \wedge T_M(\alpha,u_2,q_{20},\cdots,q_{28}) = 0\},
\end{aligned} \tag{4.38}$$

where

$$\begin{aligned}
T_S(\alpha,\beta,u_1) = \quad & q_{29}\beta - q_{30}\alpha\beta - q_{31}\alpha^2\beta + q_{32}\alpha^3\beta - q_{33}\beta^3 +\\
& q_{34}\alpha\beta^3 + u_1(-q_{35} + q_{36}\alpha - q_{37}\alpha^2 + q_{38}\beta^2 +\\
& q_{39}\alpha^3 - q_{40}\alpha\beta^2) + (-(-q_1\beta - q_2\alpha\beta + q_3a^2\beta + q_4\beta^3 +\\
& u_1(-q_5 - q_6\alpha + q_7\alpha^2 - q_8\alpha^3 + q_9\beta^2 + q_{10}\alpha\beta^2))\\
& (q_{11} - q_{12}\alpha + q_{13}\alpha^2 - q_{14}\alpha^3 + q_{15}\alpha^4 - q_{16}\beta^2 - q_{17}\alpha\beta^2 +\\
& q_{18}\alpha^2\beta^2 + q_{19}\beta^4))(-q_{41} + q\alpha - q_{42}\alpha^2 + q_{43}\beta^2 + q_{44}\alpha^3 +\\
& q_{45}\alpha\beta^2 - q_{46}\alpha^4 - q_{47}\alpha^2\beta^2 - q_{48}\beta^4).
\end{aligned} \tag{4.39}$$

Table 4.4: Aircraft uncertain coefficients

q_1	q_2	q_3	q_4	q_5	q_6	q_7
38 ∓ 0.1	170 ∓ 0.1	148 ∓ 0.1	4 ∓ 0.1	52 ∓ 0.1	2 ∓ 0.1	114 ∓ 0.1
q_8	q_9	q_{10}	q_{11}	q_{12}	q_{13}	q_{14}
79 ∓ 0.1	7 ∓ 0.1	14 ∓ 0.1	14 ∓ 0.1	10 ∓ 0.1	37 ∓ 0.1	48 ∓ 0.1
q_{15}	q_{16}	q_{17}	q_{18}	q_{19}	q_{20}	q_{21}
8 ∓ 0.1	13 ∓ 0.1	13 ∓ 0.1	20 ∓ 0.1	11 ∓ 0.1	12 ∓ 0.1	125 ∓ 0.1
q_{22}	q_{23}	q_{24}	q_{25}	q_{26}	q_{27}	q_{28}
6 ∓ 0.1	95 ∓ 0.1	21 ∓ 0.1	17 ∓ 0.1	20 ∓ 0.1	81 ∓ 0.1	139 ∓ 0.1
q_{29}	q_{30}	q_{31}	q_{32}	q_{33}	q_{34}	q_{34}
139 ∓ 0.1	112 ∓ 0.1	388 ∓ 0.1	215 ∓ 0.1	38 ∓ 0.1	185 ∓ 0.1	11 ∓ 0.1
q_{35}	q_{36}	q_{37}	q_{38}	q_{39}	q_{40}	q_{41}
35 ∓ 0.1	22 ∓ 0.1	5 ∓ 0.1	10 ∓ 0.1	17 ∓ 0.1	44 ∓ 0.1	3 ∓ 0.1
q_{42}	q_{43}	q_{44}	q_{45}	q_{46}	q_{47}	q_{48}
63 ∓ 0.1	34 ∓ 0.1	142 ∓ 0.1	63 ∓ 0.1	54 ∓ 0.1	69 ∓ 0.1	26 ∓ 0.1

The QSI algorithm generates in 1 hour on a Pentium IV 1.5GHz, and $\epsilon = 0.1$, the paving of Figure 4.13, where the red area corresponds to an inner approximation of the solution set Σ, the blue region corresponds to an inner approximation of the non solution set and the green region is undefined.

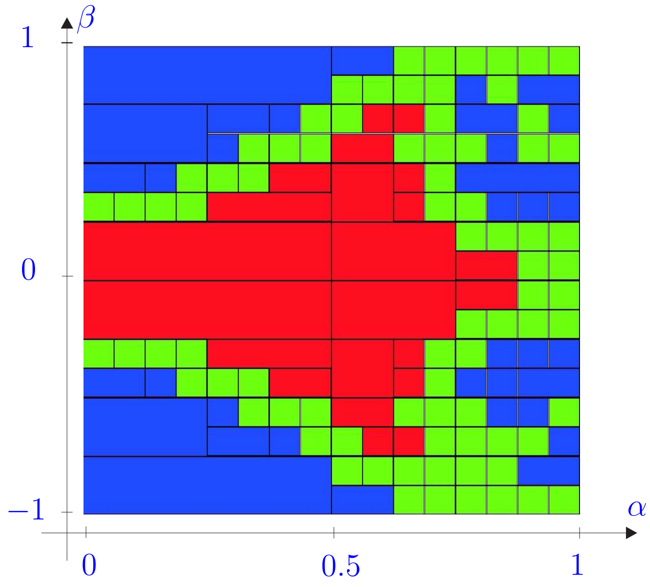


Figure 4.13: QSI output for the aircraft equilibrium problem.

Remark 4.3.3. *Details on the software implementation of the QSI algorithm (QSI solver) are explained in Chapter 8. The sources for introducing the previous examples to the QSI solver are found in Appendix A.*

■

4.4 Conclusions

In this chapter, a new algorithm for computing inner and outer approximations of the solution set of a class of QRCs is presented. This algorithm, referred to as the QSI algorithm, is based on branch-and-bound techniques and Modal Interval

Analysis. Several examples are provided together with some comparisons with the state-of-the-art techniques. From these comparisons, it is concluded that the proposed approach improves some aspects of the existing solvers. For example, the QSI algorithm can naturally deal with equality predicates while the other software implementations do not support this type of predicates. Moreover, the obtained results shows that the proposed approach can improve the computation time in some cases. This improvement is obviously related to the nature of problem and it can not be said that the proposed approach is better in general. For instance, with problems involving more than 3 free-variables, methods based on contractors (e.g. Interval Constraint Propagation) can be more efficient than the QSI algorithm. Thus, it can be concluded that the proposed technique contributes to the resolution of QRCs but, it could be improved by combining it with other techniques like Interval Constraint Propagation.

Chapter 5

Continuous Minimax Optimization Using Modal Intervals

This chapter is not devoted to the resolution of quantified real constraints but to the resolution of continuous minimax optimization problems using Modal Interval Analysis. This chapter can be seen as a collateral result of the implementation of the f^* algorithm presented in Section 3.3. More specifically, two versions of an algorithm for solving continuous minimax optimization problems are presented: One, which is devoted to solving unconstrained minimax optimization problems, and another, devoted to solve constrained minimax optimization problems.

5.1 Introduction

Many problems of practical importance can be described by mathematical models defining minimax values of certain functions. The Chebyshev Approximation problem, finding optimal strategies in Game Theory and minimizing the effect of tolerances in engineering design are classic examples. While Global Optimization has received much attention from the interval community Basso (1985); Hansen (1992); Nickel (1986) and Interval Methods are now known as a very powerful approach to dealing with this problem, only a few research studies have addressed minimax problems Jaulin (2001a); Jaulin *et al.* (2001); Sotiropoulos (2004); Zuche *et al.* (1997); Zuhe *et al.* (1990). The purpose of this chapter is to present a new interval approach to solving the next continuous minimax problem Demyanov & Malozemov (1990):

$$\begin{aligned}&f(\boldsymbol{u}^*, \boldsymbol{v}^*) = \min_{\boldsymbol{u} \in \boldsymbol{U}'} h(\boldsymbol{u}), \\ &\textit{with} \\ &h(\boldsymbol{u}) = \max_{\boldsymbol{v} \in \boldsymbol{V}'} f(\boldsymbol{u}, \boldsymbol{v}) \textit{ subject to } \boldsymbol{g}(\boldsymbol{u}, \boldsymbol{v}) \leq 0.\end{aligned} \tag{5.1}$$

where,

- $f(\boldsymbol{u}, \boldsymbol{v})$ is a $\mathbb{R}^n$ to $\mathbb{R}$ continuous function defined on the domain $\boldsymbol{X}' = (\boldsymbol{U}', \boldsymbol{V}') \subseteq \mathbb{R}^{n_{\boldsymbol{u}}} \times \mathbb{R}^{n_{\boldsymbol{v}}}$,
- $(\boldsymbol{u}^*, \boldsymbol{v}^*) \in (\boldsymbol{U}', \boldsymbol{V}')$ is a *minimax point*,
- $f(\boldsymbol{u}^*, \boldsymbol{v}^*)$ is the *minimax value* of the function on the *minimax point*,
- $\boldsymbol{g}(\boldsymbol{u}, \boldsymbol{v}) \leq 0$ is a set of inequality constraints involving $(\boldsymbol{u}, \boldsymbol{v})$.

Definition 5.1.1. *Let $f(\boldsymbol{x})$ be a $\mathbb{R}^n$ to $\mathbb{R}$ continuous function defined in an interval $\boldsymbol{X}' = (\boldsymbol{U}', \boldsymbol{V}') \in I(\mathbb{R}^n)$. A point $\boldsymbol{x}^*_{minimax} = (\boldsymbol{u}^*, \boldsymbol{v}^*) \in (\boldsymbol{U}', \boldsymbol{V}')$ is a minimax point of the problem defined by Equation 5.1 iff*

$$f(\boldsymbol{u}^*, \boldsymbol{v}^*) = \min_{\boldsymbol{u} \in \boldsymbol{U}'} \max_{\boldsymbol{v} \in \boldsymbol{V}'} f(\boldsymbol{u}, \boldsymbol{v}) \tag{5.2}$$

and

$$\boldsymbol{g}(\boldsymbol{u}^*, \boldsymbol{v}^*) \leq 0. \tag{5.3}$$

■

The problem of constrained minimax optimization consists on finding the minimax point $\boldsymbol{x}^*_{minimax}$ together with the minimax value $f(\boldsymbol{x}^*_{minimax})$ of f in $\boldsymbol{X}'$ such that the involved constraints are satisfied.

Example 5.1.1. *Given the constrained minimax optimization problem*

$$\min_{u \in [-1,1]'} h(u), \tag{5.4}$$

where

$$h(u) = \max_{u \in [-1,1]'} (v-1)^2 + u^2, \tag{5.5}$$

subject to the constraint $u^2 + v^2 - 1 \leq 0$. *Let us isolate* v *from the previous constraint*

$$v = -\sqrt{1-u^2} \tag{5.6}$$

then,

$$h(u) = (-\sqrt{1-u^2} - 1)^2 + u^2, \tag{5.7}$$

and

$$\min_{u \in [-1,1]'} (-\sqrt{1-u^2} - 1)^2 + u^2. \tag{5.8}$$

Thus, we have $u^* = 1$ *or* $u^* = -1$, *which produces*

$$f(u^*, v^*) = (-1)^2 + 1^2 = 2, \tag{5.9}$$

and consequently

$$v^* = -\sqrt{1-u^2} = 0. \tag{5.10}$$

Obviously, not all the minimax optimization problems can be analytically solved as easily as the previous problem.

■

The proposed approach for solving the stated problem in Equation 5.1 is the theory of Modal Interval Analysis (MIA) Gardeñes *et al.* (2001). This selection is motivated by the fact that the problem of finding the minimax value of a continuous function is in the core of MIA.

Remark 5.1.1. *The proposed approach to solve continuous minimax optimization problems using Modal Interval Analysis is an original contributions of this thesis. This work has been presented in an international conference* Sainz et al. *(2004) and a related paper has been submitted for publication in a journal* Sainz et al. *(2006a).*

■

5.2 Minimax optimization using modal intervals

As already mentioned in Chapter 3, a key concept of MIA is the f^* extension of a continuous function f to a modal interval vector $\boldsymbol{X}$ defined by

$$\begin{aligned} f^*(\boldsymbol{X}) &:= \bigvee_{\boldsymbol{u}\in\boldsymbol{U}'} \bigwedge_{\boldsymbol{v}\in\boldsymbol{V}'} [f(\boldsymbol{u},\boldsymbol{v}), f(\boldsymbol{u},\boldsymbol{v})] = \\ &= [\min_{\boldsymbol{u}\in\boldsymbol{U}'} \max_{\boldsymbol{v}\in\boldsymbol{V}'} f(\boldsymbol{u},\boldsymbol{v}), \max_{\boldsymbol{u}\in\boldsymbol{U}'} \min_{\boldsymbol{v}\in\boldsymbol{V}'} f(\boldsymbol{u},\boldsymbol{v})] \end{aligned} \tag{5.11}$$

where $x = (\boldsymbol{u},\boldsymbol{v})$ is the component-splitting corresponding to the proper and improper components of $X = (\boldsymbol{U},\boldsymbol{V})$, and $\vee$ and $\wedge$ indicate the interval lattice operators *meet* and *join*.

From Equation 5.11, it can be observed that the f^* extension is defined as a continuous minimax problem together with a maximin problem. This is the key property which will be used to solve the problem stated in Equation 5.1.

The following assertion can be established,

$$f(\boldsymbol{u}^*,\boldsymbol{v}^*) = \min_{\boldsymbol{u}\in\boldsymbol{U}'} \max_{\boldsymbol{v}\in\boldsymbol{V}'} f(\boldsymbol{u},\boldsymbol{v}) = Inf(f^*(\boldsymbol{U},\boldsymbol{V})). \tag{5.12}$$

where $\boldsymbol{U}$ is a proper interval, $\boldsymbol{V}$ is an improper interval and Inf is an interval operator which returns the left bound of an interval.

Example 5.2.1. *Given the continuous minimax optimization problem,*

$$f(\boldsymbol{u}^*,\boldsymbol{v}^*) = \min_{u\in[-5,5]'} \max_{v\in[-6,6]'} f(u,v), \tag{5.13}$$

where $f(u,v) = u + v$. Computing the f^* extension of f on $\boldsymbol{X} = ([-5,5],[6,-6])$ can be done by simply replacing the real variables by their associated intervals and the real operators by their intervals counterparts,

$$f^*(\boldsymbol{X}) = [-5,5] + [6,-6] = [1,-1]. \tag{5.14}$$

Then, taking the left bound of the obtained result, the minimax value is obtained,

$$f(x_1^*, x_2^*) = \min_{x_1 \in [-5,5]'} \max_{x_2 \in [-6,6]'} f(x_1, x_2) = Inf(f^*(\boldsymbol{X})) = 1. \tag{5.15}$$

Moreover, it is easy to see that the minimax value is achieved at the minimax point $(x_1^*, x_2^*) = (-5, 6)$.

■

As mentioned in Section 3.2.1.6, the f^* extension is not attainable in general by means of basic arithmetic modal interval computations and usually requires more sophisticated techniques to compute it, or simply, to approximate it.

5.2.1 The monotonic case

When certain conditions of monotonicity are accomplished, the so-called D-transformation (see Section 3.2.1.7) can be applied to exactly compute, without considering the roundings, the f^* extension. More specifically, when all the variables are totally monotonic, the following assertion can be established,

$$f(\boldsymbol{u}^*, \boldsymbol{v}^*) = \min_{\boldsymbol{u} \in \boldsymbol{U}'} \max_{\boldsymbol{v} \in \boldsymbol{V}'} f(\boldsymbol{u}, \boldsymbol{v}) = Inf(OutR(fR(\boldsymbol{U}D, \boldsymbol{V}D))). \tag{5.16}$$

where fR is an interpretable rational interval extension (see Section 3.2.1.7) and $OutR$ is an outer rounding considering a floating-point arithmetic.

Moreover, the minimax points can be obtained by applying the following rule,

$$u_i^* = \begin{cases} Inf(\boldsymbol{U}_i') & if\ \partial f/\partial u_i \geq 0, \\ Sup(\boldsymbol{U}_i') & if\ \partial f/\partial u_i \leq 0, \end{cases} \tag{5.17}$$

and

$$v_i^* = \begin{cases} Sup(V_i') & if\ \partial f/\partial v_i \geq 0, \\ Inf(V_i') & if\ \partial f/\partial v_i \leq 0, \end{cases} \tag{5.18}$$

where Inf and Sup are the interval operators which respectively return the left bound and the right bound of an interval, and i is the subindex of the variable inside its corresponding vector $\boldsymbol{u}$ or $\boldsymbol{v}$.

Example 5.2.2. *Given the continuous minimax optimization problem,*

$$f(\boldsymbol{u}^*, \boldsymbol{v}^*) = \min_{u\in[0,1]'} \max_{v\in[2,8]'} f(u,v), \tag{5.19}$$

where,

$$f(u,v) = u^2 + v^2 + 2uv - 20u - 20v + 100, \tag{5.20}$$

or

$$f(u,v) = u_1^2 + v_1^2 + 2u_2v_2 - 20u_3 - 20v_3 + 100, \tag{5.21}$$

where the subindexes represent the different multi-incidences. Then, it is possible to approximate $f^*([0,1],[8,2])$ by means of an interpretable rational interval function $fR([0,1],[8,2])$. First of all, the monotony for each variable and for each one of its incidences, considered as different variables, has to be computed:

$$\begin{aligned}
&\partial f(u,v)/\partial u = 2u + 2v - 20 \in 2*[0,1]' + 2*[2,8]' - 20 = [-16,-2]' \leq 0,\\
&\partial f(u,v)/\partial u_1 = 2u \in 2*[0,1]' = [0,2]' \geq 0,\\
&\partial f(u,v)/\partial u_2 = 2v \in 2*[2,8]' = [4,16]' \geq 0,\\
&\partial f(u,v)/\partial u_3 = -20 \leq 0,\\
&\partial f(u,v)/\partial v = 2v + 2u - 20 \in 2*[2,8]' + 2*[0,1]' - 20 = [-16,-2]' \leq 0,\\
&\partial f(u,v)/\partial v_1 = 2v \in [4,16]' \geq 0,\\
&\partial f(u,v)/\partial v_2 = 2u \in [0,2]' \geq 0,\\
&\partial f(u,v)/\partial v_3 = -20 \leq 0.
\end{aligned} \tag{5.22}$$

Notice that for computing the monotony, the range of the variables is taken into account. As f is totally monotonic with respect to the multi-incident variables (u, v), the D-transformation (see Section 3.2.1.7) can be applied. So,

$$fR(\boldsymbol{U}D, \boldsymbol{V}D) = \tag{5.23}$$

$$Dual(U_1)^2 + Dual(V_1)^2 + 2 * Dual(U_2) * Dual(V_2) - 20 * U_3 -$$

$$20 * V_3 + 100 =$$

$$[1, 0]^2 + [2, 8]^2 + 2 * [1, 0] * [2, 8] - 20 * [0, 1] - 20 * [8, 2] + 100 = [49, 4],$$

which is an optimal calculus for $f^*(\boldsymbol{X})$. Therefore,

$$f(u^*, v^*) = \min_{u \in [0,1]'} \max_{v \in [2,8]'} f(u, v) = Inf(f^*(\boldsymbol{X})) = 49, \tag{5.24}$$

Thus, by applying Equation 5.17, the minimax point is $(u^*, v^*) = (1, 2)$.

■

5.2.2 The non-monotonic case

When the monotonicity conditions are not fulfilled, only approximations of the minimax value can be achieved with rational computations. Therefore, it can be said that the minimax value is encompassed by,

$$Inf(InnR(fR(\boldsymbol{U}Dt, \boldsymbol{V}D))) \leq \min_{\boldsymbol{u} \in \boldsymbol{U}'} \max_{\boldsymbol{v} \in \boldsymbol{V}'} f(\boldsymbol{u}, \boldsymbol{v}) \leq Inf(OutR(fR(\boldsymbol{U}D, \boldsymbol{V}Dt))), \tag{5.25}$$

where D and t are the corresponding D-transformation and t-transformation (see Section 3.2.1.7) and $InnR$ and $OutR$ are respectively inner and outer roundings of a digital scale. In this case, the rule for finding the minimax point stated in Equation 5.17, can only be applied over the variables which are totally monotonic.

Example 5.2.3. *Given the Example 5.2.2 but changing the interval U to $[0, 6]$. In this case, none of the variables are totally monotonic, therefore the D-transformation* *can not be applied and instead of that, the t-transformation is applied. Thus, it is possible to obtain an outer approximation of f^* by applying the t-transformations on the variable V. The following calculus is obtained,*

$$\begin{aligned}
& fR(\boldsymbol{U}, \boldsymbol{V}t) = \\
& U_1^2 + Point(V_1)^2 + 2 * U_2 * Point(V_2) - 20 * U_3 - \\
& 20 * Point(V_3) + 100 = \\
& [0,6]^2 + [5,5]^2 + 2 * [0,6] * [5,5] - 20 * [0,6] - \\
& 20 * [5,5] + 100 = [-95, 121],
\end{aligned} \tag{5.26}$$

where Point is an operator returning a point-wise interval (e.g. the center of the interval). Notice that the obtained result is an overestimated outer approximation of the exact result $f^*(\boldsymbol{X}) = [9,4]$. Then, it is possible obtain an inner approximation for $f^*(\boldsymbol{X})$ by applying the t-transformation on the variable U.

$$\begin{aligned}
& fR(\boldsymbol{U}t, \boldsymbol{V}) = \\
& Point(U_1)^2 + V_1^2 + 2 * Point(U_2) * V_2 - 20 * Point(U_3) - \\
& 20 * V_3 + 100 = \\
& [3,3]^2 + [8,2]^2 + 2 * [3,3] * [8,2] - 20 * [3,3] - \\
& 20 * [8,2] + 100 = [121, -95],
\end{aligned} \tag{5.27}$$

which is an under-estimated approximation of the exact result $f^*(\boldsymbol{X}) = [9,4]$.

Thus, it can be said that the minimax value is encompassed between the left bounds of the two obtained approximations,

$$-95 \le \min_{u \in [0,6]'} \max_{v \in [2,8]'} f(u,v) \le 121, \tag{5.28}$$

and the minimax point lies on the box $\boldsymbol{X}' = ([0,6]', [2,8]')$*, what is not very useful from a practical point of view.*

■

It has been shown that computing the minimax value of a continuous function can be done through the computation of its f^* extension. However, good approximations of the minimax value can only be achieved under certain conditions of monotony. When these monotony conditions are not satisfied, an algorithm is provided in the next section to reduce the overestimation phenomenon.

5.3 Algorithm

As mentioned before, computing the minimax value of a continuous function f can be seen as computing (or approximating) its f^* extension. However, interval evaluations can cause an important overestimation due to possible multiple occurrences of a variable. In this section, an extension of the f^* algorithm presented in Section 3.3, which allows to approximate the minimax value of a continuous function f, is described. Moreover, a minimax optimization problem normally requires the minimax point, thus, the proposed algorithm also returns a list of boxes (interval vectors) which are candidate to contain the minimax point(s). Firstly, the unconstrained version of the minimax algorithm is presented and in a second part, the constrained version is explained.

5.3.1 Unconstrained version

The unconstrained version of the minimax is similar to the f^* algorithm presented in Section 3.3. However, as only the minimax bound of the f^* computation is required, some slight modifications are introduced to focus on the computation of this bound. These modifications consist of the following criteria.

Stopping criteria: The stopping condition is substituted by the next one

$$\textbf{While } \{Minimax(Inner, Outer) < \varphi\}, \tag{5.29}$$

where $Minimax$ is a function defined by

$$Minimax(Inner, Outer) := |Inf(Outer) - Inf(Inner)|, \tag{5.30}$$

where Inf is the left bound of the corresponding approximations.

Remark 5.3.1. *The stopping condition concerning the satisfaction of the logical formula is nonsense in the minimax optimization algorithm.*

■

Bounding criteria: The bounding criteria is substituted by

- a $Cell$ is not bisected $Inf(Inn(Cell)) \leq Inf(Out(Strip))$, because no division of any improper component $\boldsymbol{V}$ will improve the minimax approximation.

Similarly,

- a $Strip$ is not bisected when $Inf(Out(Strip)) \geq Inf(Inner)$, because no division through any proper component $\boldsymbol{U}$ will improve the minimax approximation. Moreover, this $Strip$ can be eliminated from the $StripSet$.

Selection strategy: The selection strategy is slightly modified and consists of selecting the $Strip$ and the $Cell$ with the biggest $Minimax(Inn(.), Out(.))$, and which left bound approximations $Inf(Inn(.))$ and $Inf(Out(.)))$ match at least one of these bounds with one of the left bounds of the global approximation $Inf(Inner)$ or $Inf(Outer)$.

Return value: Instead of returning the inner and outer approximations of the f^*, which is nonsense, the minimax optimization algorithm returns the minimax value $Minimax(Inner, Outer)$.

Minimax point selection: As mentioned before, a minimax optimization problem normally requires the minimax point. Algorithm 5 takes the *StripSet* resulting from applying the minimax algorithm and returns a list of boxes which are candidate to contain the minimax point(s).

Algorithm 5 SelectMinimaxBoxes

```
Input: Minimax approximation ($Minimax(Inner, Outer)$) and $StripSet$.
Output: List of candidate boxes to contain the minimax point ($MinimaxList$).
1: for $Cell$ in $Strip$ in $StripSet$ do
2:     Compute $Cell$ approximations
           $Inn(Cell) = InnR(fR(\boldsymbol{U}Dt, Prop(\boldsymbol{V})Dt)),$
           $Out(Cell) = OutR(fR(\boldsymbol{U}D, Prop(\boldsymbol{V})D)).$
3:     if $Minimax(Cell) > Minimax(Inner, Outer)$ then
4:         Eliminate $Cell$.
5:     else
6:         Enqueue $Cell$ to $MinimaxList$.
7:     end if
8: end for
9: return $MinimaxList$.
```

5.3.1.1 Examples

Several examples, some of them taken from the literature and others invented, have been solved to show the viability of the proposed algorithm. All these tests have been executed on a Pentium IV M 1.6 GHz. Notice that no comparison with other techniques has been carried out. The reason of this lack of comparisons is the absence of validated numerical software for solving continuous minimax optimization problems.

Example 5.3.1. *Given the same minimax optimization problem from Example 5.2.3.*

$$f(\boldsymbol{u}^*, \boldsymbol{v}^*) = \min_{u \in [0,1]'} \max_{v \in [2,8]'} f(u, v), \tag{5.31}$$

where,

$$f(u, v) = u^2 + v^2 + 2uv - 20u - 20v + 100. \tag{5.32}$$

Applying the proposed algorithm with $\varphi = 10^{-6}$, an $\epsilon = 10^{-6}$, the following result is obtained in 0.01 seconds.

- *Minimax : [8.999999,9.000001]'.*
- *MinimaxList : {([4.999511, 5.000107]',[1.999999, 2.000001]'); ([4.999877, 5.003174]',[7.999999, 8.000000]')}.*

■

Example 5.3.2. *Given the minimax optimization problem taken from Demyanov & Malozemov (1990)*

$$\min_{z \in [-\pi,\pi]'} \max_{y \in [-\pi,\pi]')} f(y, z), \tag{5.33}$$

where

$$f(y, z) = (\cos y + \Sigma_{k=1}^{m} \cos((k+1)y + z_k))^2 \tag{5.34}$$

For $m = 1$, $\varphi = 10^{-3}$ and $\epsilon = 10^{-3}$, the following result is obtained in 0.8 seconds.

- *Minimax : [3.098207, 3.098209]'.*
- *MinimaxList : {([-1.570800, -1.570800]',[6.336914e-01, 6.367676e-01]'); ([-1.570794, -1.570794]',[2.505542, 2.508618]'); ([1.570794, 1.570794]',[-2.508618, -2.505542]');*

([1.570800, 1.570800]',[-6.367676e-01, -6.336914e-01]')}.

■

Example 5.3.3. *Given the minimax optimization problem inspired from Hansen (1992)*

$$\min_{x_1 \in [-1,2]'} \max_{(x_2,x_3) \in ([-1,1]',[-1,1]')} f(x_1, x_2, x_3), \tag{5.35}$$

where,

$$f(x_1, x_2, x_3) = \sum_{k=1}^{10} (e^{-0.1kx_1} - e^{-0.1kx_2} - (e^{-0.1k} - e^{-k})x_3)^2 \tag{5.36}$$

For $\varphi = 10^{-3}$, $\epsilon = 10^{-3}$ *and a computation time of 9.5 seconds, the following result is obtained.*

- *Minimax : [13.800542,13.801386]'.*
- *MinimaxList :* {*([-0.343384, -0.343292]',[-1,-0.999999]',[0.999999,1]');*

  ([1.859375,2]',[0.999999,1.]'[-0.375000,-0.335937]')}.

■

5.3.2 Constrained version

The problem of finding the minimax values when, besides domain restrictions of the interval bounds, there exist other restrictions defined by inequality constraints, cannot be solved with the already presented unconstrained minimax optimization algorithm. However, by introducing some modifications to it, it is possible to tackle with the constrained case. For this purpose, each cell of the partition have to be evaluated with the involved restrictions, then, if the cell is inconsistent, it must be eliminated, while in other cases, it must be kept until subsequent divisions.

Let $\boldsymbol{\Sigma}$ be the feasible region and let $(\boldsymbol{U}'_j, \boldsymbol{V}'_{k_j})$ be a sub-box of the initial interval domain $\boldsymbol{X}' = (\boldsymbol{U}', \boldsymbol{V}')$. The following propositions must be tested,

$$(\forall \boldsymbol{u} \in \boldsymbol{U}'_j)(\forall \boldsymbol{v} \in \boldsymbol{V}'_{k_j})\ \boldsymbol{g}(\boldsymbol{u}, \boldsymbol{v}) \leq 0, \tag{5.37}$$

using the following modal interval inclusions,

$$\boldsymbol{g}R(\boldsymbol{U}_j, \boldsymbol{V}_{k_j}) \subseteq (-\infty, 0], \tag{5.38}$$

where $\boldsymbol{g} \in \mathbb{R}^n \to \mathbb{R}^m$ is a vector of continuous real functions and $\boldsymbol{U}_j$ and $\boldsymbol{V}_{k_j}$ are proper intervals. If all the inclusions are true, the cell $(\boldsymbol{U}_j, \boldsymbol{V}_{k_j})$ is consistent and belongs to the partition named $\boldsymbol{\Pi}^{(1)}$. If one or more of the next inclusions

$$\boldsymbol{g}R_i(\boldsymbol{U}_j, \boldsymbol{V}_{k_j}) \subseteq [0, \infty)(i = 1, \ldots, n), \tag{5.39}$$

is true, the cell is inconsistent, it belongs to the partition named $\boldsymbol{\Pi}^{(2)}$ and must be eliminated. Otherwise, the cell belongs to the partition named $\boldsymbol{\Pi}^{(3)}$ and it is kept.

Let $\boldsymbol{\Sigma}_{\boldsymbol{U}}$ be the projection of the feasible region $\boldsymbol{\Sigma}$ on $\boldsymbol{U}$ and let $\boldsymbol{\Sigma}_{\check{\boldsymbol{u}}}$ be the intersection of $\boldsymbol{\Sigma}$ with $\boldsymbol{u} = \check{\boldsymbol{u}}$. For a partition $\boldsymbol{\Pi}^l$ (with $l = (1), (3)$), let $(\boldsymbol{\Pi}^l_p \boldsymbol{\Pi}^l_i)$ be its split on its proper and improper components, let $r^{(l)}$ be the number of elements of $\boldsymbol{\Pi}^l_p$ and let $s^{(l)}_j$ be the number of improper boxes corresponding to the strip j in the partition $\boldsymbol{\Pi}^l_i$. Figure 5.1 illustrates the feasible region in two dimensions.

Let us define $f^*(\boldsymbol{\Sigma})$ by

$$f^*(\boldsymbol{\Sigma}) = [\min_{\boldsymbol{u} \in \boldsymbol{\Sigma}'_U} \max_{\boldsymbol{v} \in \boldsymbol{\Sigma}'_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v}), \max_{\boldsymbol{u} \in \boldsymbol{\Sigma}'_U} \min_{\boldsymbol{v} \in \boldsymbol{\Sigma}'_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v})]. \tag{5.40}$$

As

$$\min_{\boldsymbol{u} \in \boldsymbol{\Pi}^{(1)+(3)'}_U} \max_{\boldsymbol{v} \in \boldsymbol{\Pi}^{(1)'}_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v}) \leq \min_{\boldsymbol{u} \in \boldsymbol{\Sigma}'_U} \max_{\boldsymbol{v} \in \boldsymbol{\Sigma}'_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v}) \leq \min_{\boldsymbol{u} \in \boldsymbol{\Pi}^{(1)'}_U} \max_{\boldsymbol{v} \in \boldsymbol{\Pi}^{(1)+(3)'}_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v}), \tag{5.41}$$

$$\max_{\boldsymbol{u} \in \boldsymbol{\Pi}^{(1)'}_U} \min_{\boldsymbol{v} \in \boldsymbol{\Pi}^{(1)+(3)'}_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v}) \leq \max_{\boldsymbol{u} \in \boldsymbol{\Sigma}'_U} \min_{\boldsymbol{v} \in \boldsymbol{\Sigma}'_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v}) \leq \max_{\boldsymbol{u} \in \boldsymbol{\Pi}^{(1)+(3)'}_U} \min_{\boldsymbol{v} \in \boldsymbol{\Pi}^{(1)'}_{\boldsymbol{u}}} f(\boldsymbol{u}, \boldsymbol{v}), \tag{5.42}$$

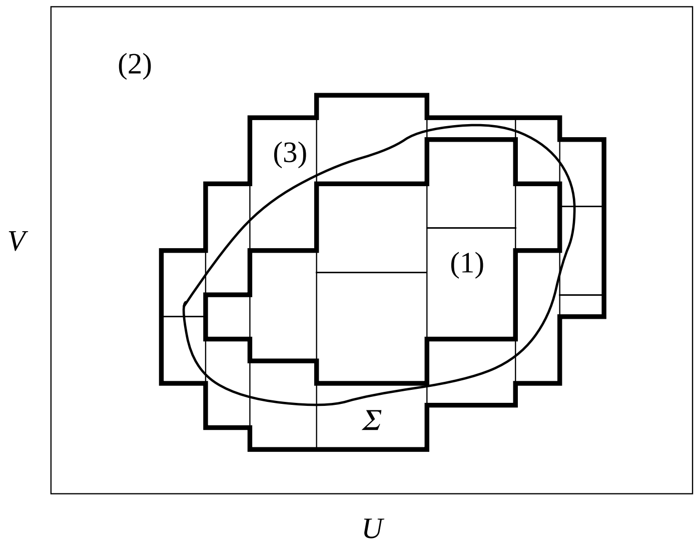


Figure 5.1: Feasibility region and partitions.

then

$$f^*(\boldsymbol{\Sigma}) \supseteq f^*(\boldsymbol{\Pi}_p^{(1)}, \boldsymbol{\Pi}_i^{(1)+(3)}) \supseteq \bigvee_{j\in\{1^{(1)},\ldots,r^{(1)}\}} \bigwedge_{k_j\in\{1_j^{(1)+(3)},\ldots,s_j^{(1)+(3)}\}} InnR(fR(\check{\boldsymbol{u}}_j, V_{k_j}) \tag{5.43}$$

$$f^*(\boldsymbol{\Sigma}) \subseteq f^*(\boldsymbol{\Pi}_p^{(1)+(3)}, \boldsymbol{\Pi}_i^{(1)}) \subseteq \Big(\bigvee_{j\in\{1^{(1)},\ldots,r^{(1)}\}} \bigwedge_{k_j\in\{1_j^{(1))},\ldots,s_j^{(1)}\}} OutR(fR(\check{\boldsymbol{u}}_j,\, \boldsymbol{V}_{k_j}))) \;\vee\; E \tag{5.44}$$

where

$$E = \bigvee_{j\in\{1^{(3)},\ldots,r^{(3)}\}-\{1^{(1)},\ldots,r^{(1)}\}} \bigvee_{k_j\in\{1_j^{(3)},\ldots,s_j^{(3)}\}} OutR(fR(\check{\boldsymbol{U}}_j, Dual(\boldsymbol{V}_{k_j}))). \tag{5.45}$$

In accordance with the previous reasoning, the necessary additional (or substitutive) steps for solving the constrained version of the minimax optimization problem, with respect to the unconstrained version, are summarized with the following steps.

- For each $Cell$, define its consistency, that is, if $(Cell\ in\ \boldsymbol{\Pi}^{(1)})$, $(Cell\ in\ \boldsymbol{\Pi}^{(2)})$ or $(Cell\ in\ \boldsymbol{\Pi}^{(3)})$.

- Compute inner and outer approximations of the resulting *Cell* partitions, as follows.
  If $(Cell \in \boldsymbol{\Pi}^{(1)})$ then,
  $$Inn(Cell^{(1)}) = InnR(fR(\boldsymbol{U}Dt, \boldsymbol{V}D)), \tag{5.46}$$
  $$Out(Cell^{(1)}) = OutR(fR(\boldsymbol{U}D, \boldsymbol{V}Dt)). \tag{5.47}$$
  Else, if $(Cell \in \boldsymbol{\Pi}^{(3)})$ then,
  $$Inn(Cell^{(3)}) = InnR(fR(\boldsymbol{U}Dt, \boldsymbol{V}D)), \tag{5.48}$$
  $$Out(Cell^{(3)}) = OutR(fR(\boldsymbol{U}D, Dual(\boldsymbol{V})D)). \tag{5.49}$$
  It is important to remark that, in order to compute the inner approximation $InnR(fR(\boldsymbol{U}Dt, \boldsymbol{V}D))$, the D-transformation over a variable of a $\boldsymbol{U}$ partition (*Strip*), requires the variable to be totally monotonic along the $\boldsymbol{U}$ partition.

- Compute inner and outer approximations of *Strip*, that is
  $$Inn(Strip) = \bigwedge_{\{Cell^{(1,3)}\ in\ Strip\}} Inn(Cell^{(1,3)}), \tag{5.50}$$
  $$Out(Strip) = \bigwedge_{\{Cell^{(1)}\ in\ Strip\}} Out(Cell^{(1)}). \tag{5.51}$$
  where $Cell^{(1,3)}$ is a *Cell* belonging to the partitions $\boldsymbol{\Pi}^{(1)}$ or $\boldsymbol{\Pi}^{(3)}$.

- Compute global inner and outer approximations, that is
  $$Inner = \bigvee_{\{Strip^{(1)}\ in\ StripSet\}} Inn(Strip^{(1)}), \tag{5.52}$$
  $$Outer = \bigvee_{\{Strip^{(1,3)}\ in\ StripSet\}} Out(Strip^{(1,3)}). \tag{5.53}$$
  where $Strip^{(1)}$ is a *Strip* containing at leat one $Cell^{(1)}$ and $Strip^{(3)}$ is a *Strip* not containing any $Cell^{(1)}$ and containing at leat one $Cell^{(3)}$.

- Concerning the bisection strategy, do not bisect a *Cell* or a *Strip* partition only if the objective function is totally monotonic with respect to its components $(\boldsymbol{U}, \boldsymbol{V})$ and it is consistent with respect to the involved constraints.

Remark 5.3.2. *The complexity, completeness, termination and soundness properties of the minimax optimization algorithm (both versions) are the same than for the f^* algorithm from Section 3.3.*

■

5.3.2.1 Examples

Example 5.3.4. *Consider the constrained minimax optimization problem from Example 5.1.1. For $\varphi = 10^{-5}$ and $\epsilon = 10^{-5}$, the following result is obtained in 10 seconds.*

- *Minimax : [1.997236e+00, 2.003907e+00]'.*
- *MinimaxList : {([-1.000977e+00, -9.999962e-01]',[-1.956940e-03, 1.381874e-03]'); ([9.999962e-01, 1.000977e+00]',[-1.956940e-03, 1.381874e-03]')}.*

■

Example 5.3.5. *Given the constrained minimax optimization problem*

$$\min_{x_1 \in [0,6]'} h(x_1), \tag{5.54}$$

where

$$h(x_1) = \max_{x_2 \in [2,8]'} x_1^2 + x_2^2 + 2x_1x_2 - 20x_1 - 20x_2 + 100, \tag{5.55}$$

subject to the constraints

$$\begin{aligned} g_1(x_1, x_2) &= (x_1 - 5)^2 + (y - 3)^2 - 4 \geq 0, & (5.56)\\ g_2(x_1, x_2) &= (x_1 - 5)^2 + (y - 3)^2 - 16 \leq 0. & (5.57)\end{aligned}$$

For $\varphi = 10^{-5}$ and $\epsilon = 10^{-5}$, the following result is obtained in 0.17 seconds.

- *Minimax* : *[1.10255, 1.10256]'.*
- *MinimaxList* : $\{$*([4.14290, 4.14293]',[4.80701, 4.80710]');*

  ([4.14290, 4.14293]',[6.90709, 6.90714]')$\}$.

■

Example 5.3.6. *Given the constrained continuous minimax optimization problem inspired from Demyanov & Malozemov (1990)*

$$\min_{x\in[-3.14,3.14]'} h(x), \tag{5.58}$$

where

$$h(x) = \max_{y\in[-3.14,3.14]'} (cos(y) + cos(2y + x))^2, \tag{5.59}$$

subject to the constraints

$$g_1(x, y) = y - x(x + 6.28) \leq 0, \tag{5.60}$$

$$g_2(x, y) = y - x(x - 6.28) \leq 0 \tag{5.61}$$

For $\epsilon = 10^{-6}$ *and* $\varphi = 10^{-6}$*, the following result is obtained in 0.3 seconds.*

- *Minimax* : *[8.586377e-03, 8.586666e-03]'.*
- *MinimaxList* : $\{$*([-0.4370827,-0.4370812]'; [-2.553836;-2.553830]');*

  ([-0.4370827,-0.4370812]'; [-3.140000,-2.747500]')$\}$.

■

Remark 5.3.3. *Details on the software implementation of the Minimax algorithm (the MINIMAX solver) are explained in Chapter 8. The sources for introducing the previous examples to the MINIMAX solver are found in Appendix A.*

■

5.4 Conclusions

In this chapter, a new approach to deal with unconstrained and constrained continuous minimax optimization problems, and based on Modal Interval Analysis, has been introduced. The proposed methodology is based on the computation of the so-called *-semantic extension of a continuous function. Modal Interval Analysis allows to compute this extension efficiently and to obtain guaranteed results. In some simple cases, the results are obtained with simple modal interval arithmetic computations. Nevertheless, in many cases, a branch-and-bound algorithm is required to obtain tight results. This algorithm and some illustrative examples have been presented.

Chapter 6

Application to Fault Detection

In this chapter, the problem of detecting faults in dynamic systems using Modal Interval Analysis, originally presented in the thesis of Armengol Armengol (1999), is stated as a Quantified Real Constraint (QRC) (see Chapter 2). This approach is based on the principle of *analytical redundancy* and takes uncertainty into account by means of interval parameters and interval measurements. For its resolution, techniques presented in Chapter 3 are applied. My main contribution in fault detection have been: A new formulation of the original approach proposed by Armengol, a complete software implementation of the fault detection technique and the validation of the approach with academic and industrial processes. For instance, this tool has been applied to the detection of faults in different processes in the context of the European project CHEM CHEM Consortium (2000). Moreover, several papers presenting the obtained results have been published in different conferences and journals Armengol *et al.* (2003, 2004); Sainz *et al.* (2002c).

6.1 Introduction

A fault is a malfunction in a system and may have consequences like economical losses derived from lower efficiency of the system or danger for the people or the environment. Many different techniques have been developed in the recent years which intend to detect and diagnose faults. These techniques can be classified in different ways Balakrishnan & Honavar (1998); Venkatasubramanian *et al.* (2003). For instance, one distinction can be made between model-based techniques and techniques based on other kinds of knowledge like heuristic approaches, statistical approaches, learning systems, artificial neural networks, etc. Two main research communities work on model-based techniques: the FDI (Fault Detection and Isolation) community, formed by researchers with a background in control systems engineering, and the DX (Principles of Diagnosis) community, formed by researchers with a background in computer science and intelligent systems. The collaboration between these two communities in order to develop more powerful tools for fault detection and diagnosis has been one of the goals of the European Network of Excellence on Model Based Systems and Qualitative Reasoning (MONET) MONET (2006), particularly of the Bridge task group. Among the techniques developed by the FDI research community, there are classical techniques like state observers, parity equations and parameter estimation Blanke *et al.* (2003); Patton *et al.* (2000). A method to detect faults consists in comparing the behaviour of an actual system and a reference system, given by an analytical model. This principle is called *analytical redundancy.*

6.2 Analytical redundancy

Given a system model, representing an actual system or a part of it, described by the following nonlinear discrete-time equation,

$$\boldsymbol{y}(k) = \boldsymbol{f}(\boldsymbol{y}(k-1), \boldsymbol{u}(k-1), \boldsymbol{p}), \tag{6.1}$$

where $\boldsymbol{y}(k) \in \mathbb{R}^{n_y}$ and $\boldsymbol{y}(k-1) \in \mathbb{R}^{n_y}$ are the outputs of the system at instant k and $k-1$, $\boldsymbol{f}$ is a vector of continuous functions, $\boldsymbol{u}(k-1) \in \mathbb{R}^{n_u}$ is a vector of inputs at instant $k-1$ and $\boldsymbol{p} \in \mathbb{R}^{n_p}$ is a vector of parameters.

An analytical redundancy relation (ARR) is an algebraic constraint deduced from the system model which contains only measured variables. An ARR for Equation 6.1 is

$$\boldsymbol{y}_m(k) = \boldsymbol{y}_r(k), \tag{6.2}$$

where $\boldsymbol{y}_m(k)$ is the measured output of the system at instant k and $\boldsymbol{y}_r(k)$ is an analytical output of the system at instant k and computed as

$$\boldsymbol{y}_r(k) := \boldsymbol{f}(\boldsymbol{y}_m(k-1), \boldsymbol{u}(k-1), \boldsymbol{p}), \tag{6.3}$$

An ARR is used to check the consistency of the observations with respect to the system model. Therefore, a fault is detected when

$$\boldsymbol{y}_r(k) \neq \boldsymbol{y}_m(k), \tag{6.4}$$

or equivalently

$$\mathbf{r} = \boldsymbol{y}_r(k) - \boldsymbol{y}_m(k) \neq 0, \tag{6.5}$$

where $\mathbf{r}$ is called the residual of the ARR.

The main problem is that the measured output $\boldsymbol{y}_m(k)$ and the computed output $\boldsymbol{y}_r(k)$ are seldom the same because the model is, by definition, inaccurate, i.e. it is an approximate representation of the system. This is the consequence of the uncertainties of the system and the procedure of systems' modelling.

Therefore, the uncertainty of the system has to be considered. It can be taken into account when the comparison between the behaviour of the actual system and the one of its model is performed. In this case, a fault is indicated when the difference is larger than a threshold:

$$\mathbf{r} = |\boldsymbol{y}_r(k) - \boldsymbol{y}_m(k)| > \epsilon, \tag{6.6}$$

An important difficulty now is to determine the size of the threshold ϵ. If it is too small, faults are indicated even when they do not exist. These are false alarms. On the other side, if the threshold is too large, the amount of missed alarms increases.

In order to overcome this problem, the uncertainties of the system can be taken into account during the modelling procedure by means of interval parameters and interval measurements, which can be obtained by replacing the real variables (measurements and parameters) and real arithmetic operators by its interval counterparts. Then, applying the principle of analytical redundancy, but taking into account the uncertainty, a fault is detected when

$$(\forall \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\forall \boldsymbol{y}_r(k) \in \boldsymbol{Y}'_r(k))\boldsymbol{y}_r(k) - \boldsymbol{y}_m(k) \neq 0. \tag{6.7}$$

On the other hand, meanwhile the previous logical statement is not true, nothing can be assured.

6.2.1 Consistency test

Firstly, consider the consistent case, which means that the output of the model is coherent with the measured output. This assertion is expressed with the next logical statement,

$$(\exists \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\exists \boldsymbol{y}_r(k) \in \boldsymbol{Y}'_r(k))\boldsymbol{y}_r(k) - \boldsymbol{y}_m(k) = 0, \tag{6.8}$$

or equivalently,

$$\begin{aligned}&(\exists \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\exists \boldsymbol{y}_m(k-1) \in \boldsymbol{Y}'_m(k-1))(\exists \boldsymbol{u}_m(k-1) \in \boldsymbol{U}'_m(k-1))\\&(\exists \boldsymbol{p} \in \boldsymbol{P}')f(\boldsymbol{y}_m(k-1), \boldsymbol{u}(k-1), \boldsymbol{p}) - \boldsymbol{y}_m(k) = 0,\end{aligned} \tag{6.9}$$

which can be proved using the *modal interval inclusion* test defined in Section 3.2.1.8. Thus, the next implication is true

$$\begin{aligned}&Outer(\mathbf{g}^*(\boldsymbol{Y}_m(k), \boldsymbol{Y}_m(k-1), \boldsymbol{U}_m(k-1), \boldsymbol{P})) \subseteq [0,0] \Rightarrow\\&(\exists \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\exists \boldsymbol{y}_m(k-1) \in \boldsymbol{Y}'_m(k-1))(\exists \boldsymbol{u}_m(k-1) \in \boldsymbol{U}'_m(k-1))\\&(\exists \boldsymbol{p} \in \boldsymbol{P}')\mathbf{g}(\boldsymbol{y}_m(k), \boldsymbol{y}_m(k-1), \boldsymbol{u}_m(k-1), \boldsymbol{p}) = 0,\end{aligned} \tag{6.10}$$

where $\boldsymbol{Y}_m(k)$, $\boldsymbol{Y}_m(k-1)$, $\boldsymbol{U}_m(k-1)$ and $\boldsymbol{P}$ are improper intervals and $Outer(g^*)$ is an outer approximation of the *-semantic extension of the continuous function

$$\mathbf{g}(\boldsymbol{y}_m(k), \boldsymbol{y}_m(k-1), \boldsymbol{u}_m(k-1), \boldsymbol{p}) = f(\boldsymbol{y}_m(k-1), \boldsymbol{u}(k-1), \boldsymbol{p}) - \boldsymbol{y}_m(k). \tag{6.11}$$

Remark 6.2.1. *Because of the considered uncertainty, proving that the previous consistency test is true does not guarantee that any fault exists in the actual process.*
■

Referring to the inconsistent case, which means that the output of the model is not coherent with the measured output. This assertion is through the next logical statement,

$$\begin{aligned}&\neg((\exists \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\exists \boldsymbol{y}_r(k) \in \boldsymbol{Y}'_r(k))\boldsymbol{y}_r(k) - \boldsymbol{y}_m(k) = 0) \Leftrightarrow\\&(\forall \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\forall \boldsymbol{y}_r(k) \in \boldsymbol{Y}'_r(k))\boldsymbol{y}_r(k) - \boldsymbol{y}_m(k) \neq 0,\end{aligned} \tag{6.12}$$

which can also be proven using the *modal interval inclusion test* as follows,

$$\begin{aligned}&[0,0] \nsubseteq Outer(\mathbf{g}^*(\boldsymbol{Y}_m(k), \boldsymbol{Y}_m(k-1), \boldsymbol{U}_m(k-1), \boldsymbol{P})) \Rightarrow \\ &(\forall \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\forall \boldsymbol{y}_m(k-1) \in \boldsymbol{Y}'_m(k-1))(\forall \boldsymbol{u}_m(k-1) \in \boldsymbol{U}'_m(k-1)) \\ &(\forall \boldsymbol{p} \in \boldsymbol{P}') g(\boldsymbol{y}_m(k), \boldsymbol{y}_m(k-1), \boldsymbol{u}_m(k-1), \boldsymbol{p}) \neq 0, \end{aligned} \tag{6.13}$$

where $\boldsymbol{Y}_m(k)$, $\boldsymbol{Y}_m(k-1)$, $\boldsymbol{U}_m(k-1)$ and $\boldsymbol{P}$ are proper intervals.

Therefore, the problem of fault detection has been stated as a QRC satisfaction problem, which can be solved using tools provided in Chapter 3.

Remark 6.2.2. *Notice that the logic statement involved in Equation 6.13 only involves the universal quantifier* $\forall$. *For this reason, it can be solved using the classic Interval Analysis theory Moore (1966) and the use of Modal Interval Analysis is not strictly required. However, by using the proposed technique in Chapter 3, it is to tackle with the dependency problem of the interval variables in an efficient and elegant way. It is also interesting to remark that proving the consistency test of Equation 6.12 could be done by means of interval constraint propagation Benhamou & Older (1997) as proposed by Stancu in Stancu & Quevedo (2005).*
■

6.2.2 Window consistency

The *window consistency* allows to determine the consistency of a set of system measurements (inputs and output) between two sampling times with respect to the reference behavior in the same time interval. The distance between the two considered sampling times is called *window length* and is denoted by w. Then, for a window length w, a system is faulty if

$$(\forall \boldsymbol{y}_m(k) \in \boldsymbol{Y}'_m(k))(\forall \boldsymbol{y}_r(k) \in \boldsymbol{Y}'_r(k|k-w))\ \boldsymbol{y}_r(k|k-w) - \boldsymbol{y}_m(k) \neq 0, \tag{6.14}$$

where,

$$\boldsymbol{y}_r(k|k-w) = f(\boldsymbol{y}_m(k-w), u(k-1), \ldots, u(k-w), \boldsymbol{p}), \tag{6.15}$$

is the corresponding reference behavior model for a window length of w.

Notice that, for different window lengths, it can happen that with w_1 the fault is not detected but with w_2 the fault is detected. In this case, it can be assured that there is a fault because detecting with one window length is a sufficient condition to do so. Consequently, there is a fault if

$$\begin{aligned}
&(\forall \boldsymbol{y}_m(k) \in \boldsymbol{Y'}_m(k))(\forall \boldsymbol{y}_r(k) \in \boldsymbol{Y'}_r(k))\ \boldsymbol{y}_r(k) - \boldsymbol{y}_m(k) \neq 0 \qquad (6.16)\\
&\vee \ldots \vee\\
&(\forall \boldsymbol{y}_m(k) \in \boldsymbol{Y'}_m(k))(\forall \boldsymbol{y}_r(k|k-w) \in \boldsymbol{Y'}_r(k|k-w))\ \boldsymbol{y}_r(k|k-w) - \boldsymbol{y}_m(k) \neq 0.
\end{aligned}$$

The fault detection results obtained using several window lengths are generally better, (i.e. there are less missed alarms), than the ones obtained using a single window length, whatever is the length in the latter case. However, deciding which is the best window length still remains an open problem. This choice not only depends on the dynamics of the system but also on the type of fault to be detected.

Example 6.2.1. *Given the ARR corresponding to a discretized first-order model,*

$$y_r(k) = a y_m(k-1) + b u(k-1), \tag{6.17}$$

where a and b are model parameters. The corresponding ARRs for the set of window lengths $\mathbf{w} = \{1, 2, 5\}$,

$$\begin{aligned}
&\textit{Length} = 1: && y_r(k) = a y_m(k-1) + b u(k-1), && (6.18)\\
&\textit{Length} = 2: && y_r(k) = a(a y_m(k-2) + b u(k-2)) + b u(k-1), && (6.19)\\
&\textit{Length} = 5: && y_r(k) = a(a(a(a(a y_m(k-5) + b u(k-5)) + && \\
& && b u(k-4)) + b u(k-3)) + b u(k-2)) + b u(k-1). && (6.20)
\end{aligned}$$

Thus, the corresponding QRC is

$$\begin{aligned}
Length = 1: \quad & (\forall y_m(k) \in Y'_m(k))(\forall y_m(k-1) \in Y'_m(k-1)) && (6.21)\\
& (\forall u(k-1) \in U'(k-1))(\forall a \in A')(\forall b \in B') \\
& a y_m(k-1) + b u(k-1) - y_m(k) \neq 0 \\
& \vee \\
Length = 2: \quad & (\forall y_m(k) \in Y'_m(k))(\forall y_m(k-1) \in Y'_m(k-1)) && (6.22)\\
& (\forall u(k-1) \in U'(k-1))(\forall u(k-2) \in U'(k-2))(\forall a \in A') \\
& (\forall b \in B')\ a(a y_m(k-2) + b u(k-2)) + b u(k-1) - y_m(k) \neq 0 \\
& \vee \\
Length = 5: \quad & (\forall y_m(k) \in Y'_m(k))(\forall y_m(k-1) \in Y'_m(k-1)) && (6.23)\\
& (\forall u(k-1) \in U'(k-1))(\forall u(k-2) \in U'(k-2))) \\
& (\forall u(k-3) \in U'(k-3))(\forall u(k-4) \in U'(k-4) \\
& (\forall u(k-5) \in U'(k-5))(\forall a \in A')(\forall b \in B') \\
& a(a(a(a(a y_m(k-5) + b u(k-5)) + b u(k-4)) + \\
& b u(k-3)) + b u(k-2)) + b u(k-1) - y_m(k) \neq 0.
\end{aligned}$$

Therefore, proving that the previous QRC is true is equivalent to say that a fault is detected.

■

6.2.3 Fault detection algorithm

The proposed fault detection algorithm requires from the user: a process model, the process data, the uncertainty associated to the process model and process data, the window lengths $\{w_1, \ldots w_n\}$ and a time (TimeOut) to limit the computations carried out between two sample times. First of all, the algorithm builds the corresponding ARRs for each window length $\{\mathbf{g}_{w_1} = 0, \ldots, \mathbf{g}_{w_n} = 0\}$. As the necessary computing effort to deal with a larger ARR $\mathbf{g}_{w_n} = 0$ is bigger than for a shorter ARR $\mathbf{g}_{w_1} = 0$, when $w_n > w_1$, the algorithm starts, at each time point, using the shortest window length and stops when a fault is detected, thus saving computing effort and minimizing the rate of missed alarms. For each ARR, the f^* algorithm (see Section 3.3) is called to approximate the corresponding *-extension ($\mathbf{g}^*_{w_i}$). The f^* execution stops when $[0,0] \nsubseteq Outer(\mathbf{g}^*_{w_i})$, because a fault is detected or when the ARR is consistent, that is $[0,0] \subseteq Inner(\mathbf{g}^*_{w_i})$. The algorithm returns "Faulty" if at least one of the ARRs is proven to be inconsistent and returns "Perhaps" if none of the ARRs is proven to be inconsistent. This algorithm is summarized in Algorithm 6.

Algorithm 6 Fault detection algorithm

```
Input: Process model, process data, uncertainties, window lengths $\{w_1, \dots w_n\}$
    and Timeout.
Output: Faulty.
 1: Faulty=perhaps,
 2: Build the ARRs for each window length $\{\mathbf{g}_{w_1} = 0, \dots \mathbf{g}_{w_n} = 0\}$.
 3: while Available process data do
 4:   Read process data and assign it to the corresponding interval ARRs.
 5:   for i=1 to i=n do
 6:     Approximate $\mathbf{g}^*_{w_i}$ till $[0,0] \nsubseteq Outer(\mathbf{g}^*_{w_i})$ or $[0,0] \subseteq Inner(\mathbf{g}^*_{w_i})$ or Time-
        out reached.
 7:     if $[0,0] \nsubseteq Outer(\mathbf{g}^*_{w_i})$ then
 8:       Faulty=true.
 9:       Break.
10:     end if
11:   end for
12: end while
13: return Faulty.
```

Remark 6.2.3. *Notice that the Algorithm 6 does not report a fault when it is inconclusive because the timeout is reached. This is because the proposed approach prioritizes to avoid false alarms to missed alarms.*
■

6.2.4 Graphical output

The presented fault detection algorithm has been implemented under a software tool called the SQUALTRACK solver (Semi-Qualitative Tracking) in the context of the CHEM project CHEM Consortium (2000) and is currently being applied to academic and actual processes Armengol *et al.* (2003, 2004). For more details on its implementations see Chapter 8.

In order to provide a more friendly output to the final user, the SQUALTRACK solver plots to the graphical user interface (GUI) the computed inner and an outer approximations of the model output together with the measured output of the system. As all these signals are represented by interval trajectory (or envelopes), by observing if the outer approximation intersects or not with the actual measured output, it is possible to determine, in a visual way, if the process is behaving normally or faulty. However, the software does this consistency test in an automatic way.

Figure 6.1 shows the graphical output of the SQUALTRACK solver. The upper graph shows the approximations (inner in green and outer in red) for the output variable and the corresponding measurement (in black). Note that often inner and outer approximations are not graphically distinguishable because they are very close. The green bars graph in the middle indicates the longest window length that has been used at each time step. Finally, the lower graph shows a red bar when a fault is detected.

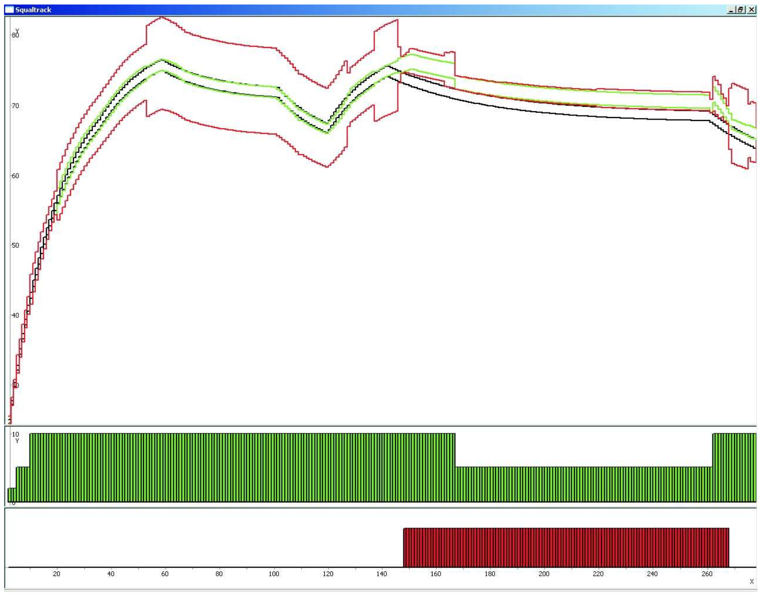


Figure 6.1: SQUALTRACK solver graphical output.

6.3 Applications

This section is devoted to describe some applications within the European project CHEM CHEM Consortium (2000). The aim of this project was to develop and implement an advanced Decision Support System (DSS) for process monitoring, data and event analysis, and operation support in industrial processes. The system was intended to be a synergistic integration of innovative software tools, which would improve the safety, product quality and operation reliability as well as reduce the economic losses due to faulty states, mainly in chemical and petrochemical processes. Among these applications there are the flexible chemical pilot plant PROCEL, owned by the Universitat Politècnica de Catalunya and situated in Barcelona (Catalonia, Spain), the steam generator pilot plant owned by the Laboratoire d'Automatique et d'Informatique Industrielle de Lille (France) and the FCC (Fluid Catalytic Cracking) plant owned by the French Institute of Petroleum (IFP) and situated in Lyon (France).

6.3.1 PROCEL pilot plant

PROCEL is constituted by three tank reactors, three heat exchangers and the necessary pumps and valves to allow changes in the configuration. The equipment of PROCEL is fully connected and the associated instrumentation allows the change of configuration by software. Figure 6.2 shows a flowsheet of PROCEL.

6.3.1.1 Testing scenarios

Three faulty scenarios that have been considered affect to reactor 1:

1. An additional input flow. The fault consists in opening the valve $VE2$ during a time period to simulate an additional input flow which is not taken into account, i.e. a perturbation.
2. A leakage. Valve $VE4$ is opened during a time period.
3. Resistor 1 ($R1$) shutdown.

For scenarios 1 and 2 a model obtained from the mass balance of reactor 1 is enough. The monitored variable is the volume of reactor 1. For scenario 3 a model obtained from the energy balance is needed. The monitored variable is the temperature of reactor 1. In both models, the values of the variables (measurements and parameters) are considered uncertain and represented by intervals. This interval values have been provided by process expert from the Universitat Politècnica de Catalunya.

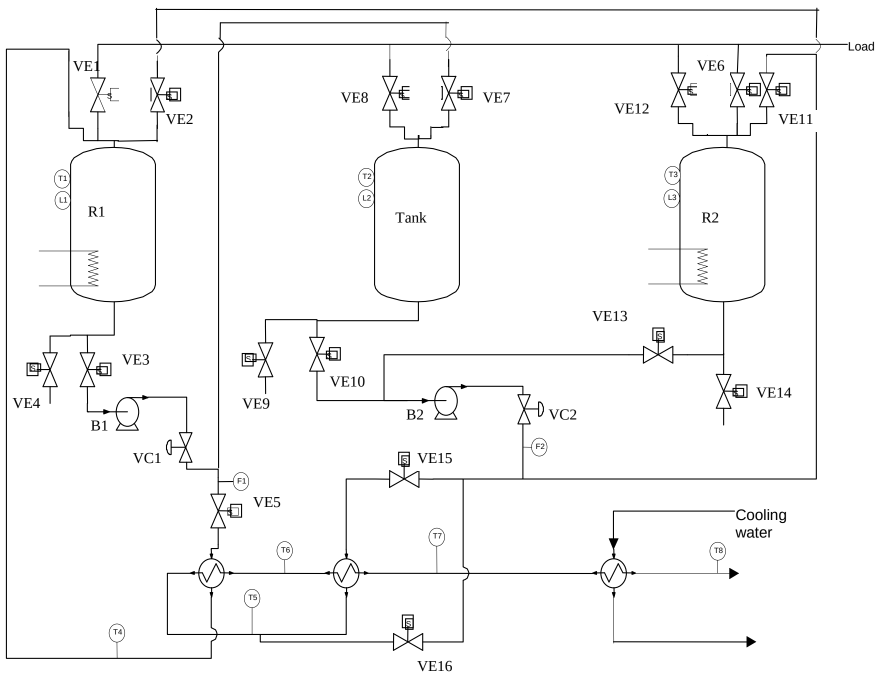


Figure 6.2: Flowsheet of PROCEL.

6.3.1.2 Mass balance model

The volume variation inside the reactor 1 is:

$$\frac{dv_{R1}}{dt} = (f_1 + f_3) - f_4 \tag{6.24}$$

where d_{R1} is the volume of liquid in the reactor, $f1$ and $f3$ are the volumetric input flows and $f4$ is the volumetric output flow. It is assumed that there is a relative error of 3 % for v_{R1} and 5 % for $f1$, $f3$ and $f4$. The corresponding discrete-time equation is

$$v_{R1}(k) = v_{R1}(k-1) + (f_1(k) + f_3(k) - f_4(k))t_s, \tag{6.25}$$

where ts is the sample time.

As v_{R1} is directly measurable, the analytical redundancy relation consists of comparing the computed value of v_{R1} with its measurement.

$$v_{R1}(k) - v_{R1_m}(k) = 0, \tag{6.26}$$

where $v_{R1_m}(k)$ is the measured volume.

6.3.1.3 Energy balance model

The temperature variation inside the reactor 1 is:

$$\begin{aligned}\frac{dt_{R1}}{dt} &= \frac{f_1\,(t_1 - t_{R1})}{v_{R1}} + \frac{f_2\,(t_2 - t_{R1})}{v_{R1}} + \dots \\ &+ \frac{f_3\,(t_3 - t_{R1})}{v_{R1}} + \frac{p_H - g_{R1}\,(t_{R1} - t_{amb})}{\rho_{R1} c_{pR1} v_{R1}}\end{aligned} \tag{6.27}$$

where t_{R1} is the temperature of the liquid in the reactor, hence the output of this subsystem. The inputs are the flows f_1, f_2 and f_3, and the corresponding temperatures t_1, t_2 and t_3. It is assumed that there is a relative error of 5 % for t_{R1}, t_1, t_2 and t_3.

The parameters of the model are: the volume of liquid v_{R1}, the ambient temperature t_{amb} ($t_{amb} \in [18, 20]' + 273.15$ K), the heater power p_H (with a relative error of 10 %), the thermal conductance of the wall g_{R1} (around 0.01 $\frac{W}{K}$), the density of the liquid ρ_{R1} (water) and its specific heat c_{pR1}.

The corresponding discrete-time equation is:

$$\begin{aligned} t_{R1}(k) &= t_{R1}(k-1) + \ldots \\ +t_s\left(\frac{1}{v_{R1}(k)}\right. & \left(f_1(k)\left(t_1(k) - t_{R1}(k-1)\right) + \ldots\right. \\ & + f_2(k)\left(t_2(k) - t_{R1}(k-1)\right) + \ldots \\ & \left.+ f_3(k)\left(t_3(k) - t_{R1}(k-1)\right)\right) + \ldots \\ & \left.+\frac{p_H(k) - g_{R1}\left(t_{R1}(k-1) - t_{amb}\right)}{\rho_{R1} c_{pR1} v_{R1}(k)}\right), \end{aligned} \qquad (6.28)$$

where t_s is the sampling time.

As $t_{R1}(k)$ is directly measurable, the analytical redundancy relation consists on comparing the simulated value of $t_{R1}(k)$ with its measurement.

$$t_{R1}(k) - t_{R1_m}(k) = 0, \qquad (6.29)$$

where $t_{R1_m}(k)$ is the measured temperature.

6.3.1.4 Testing results

Figures 6.3, 6.4 and 6.5 show the graphical window of the fault detection software when it is used to detect faults for each of the tested scenarios. Windows of lengths 5, 50, 75 and 100 samples are used for the first scenario, lengths of 10, 50, 100 and 200 for the second scenario and lengths of 10 and 25 for the third scenario. In all the graphs, time is given in samples and the sample time is 3 s.

For scenario 1 (additional input flow), the fault begins at sample 55 and ends at sample 73. It is detected from sample 77 until sample 93 and from 120 to 130. See Figure 6.3.

For scenario 2 (leakage), the fault begins at sample 272 and ends at sample 343. It is intermittently detected from sample 326 until sample 441. See Figure 6.4.

For scenario 3 (resistor 1 shutdown), the fault begins at sample 581 and ends at sample 617. It is detected from sample 610 until sample 630. See Figure 6.5.

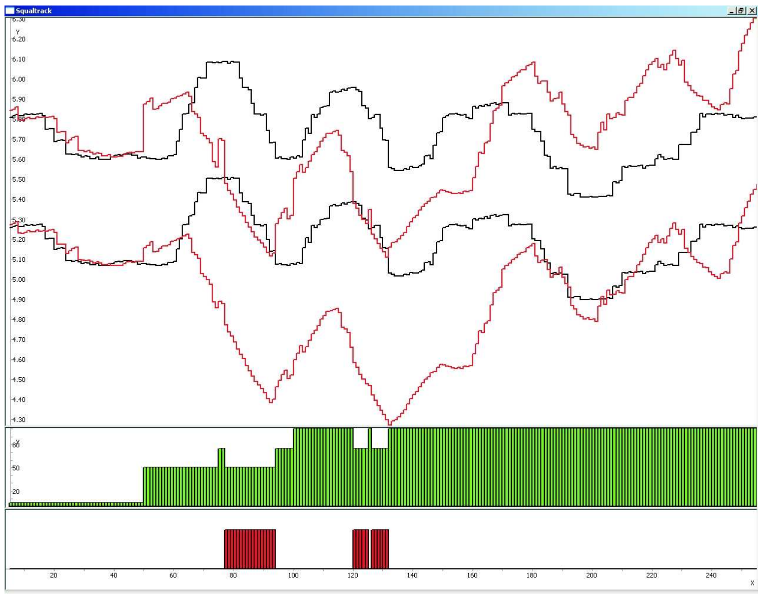


Figure 6.3: Additional input flow to reactor 1.

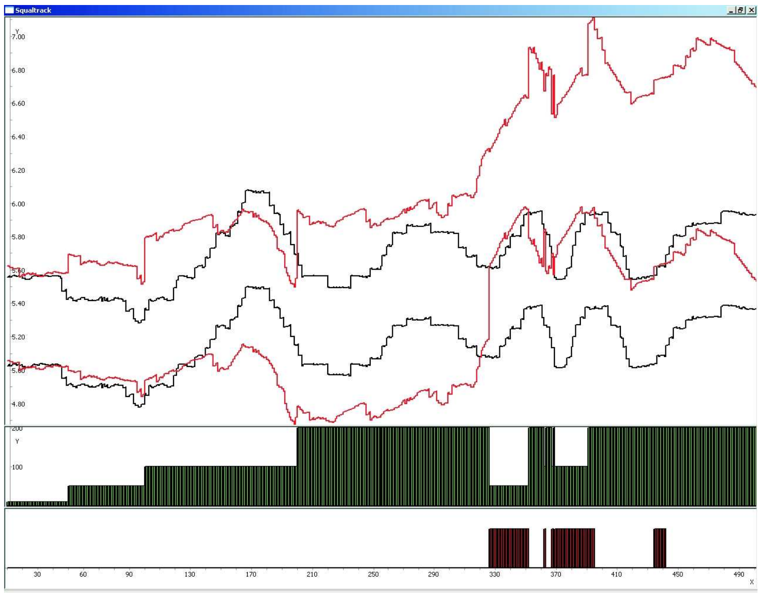


Figure 6.4: Reactor 1 leakage.

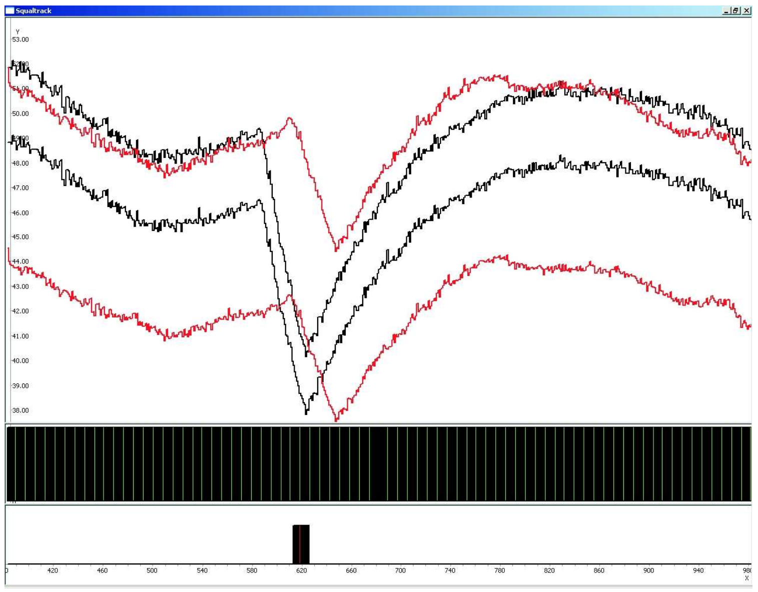


Figure 6.5: Reactor 1 heater shutdown.

6.3.2 Steam Generator pilot plant

6.3.2.1 Process description

The Steam Generator is a scale-model of a power plant. It is a complex non linear system which reproduces the same thermodynamic phenomena as the actual industrial process. As shown in the flowsheet of Figure 6.6, this installation is constituted of four main subsystems: a receiver with the feedwater supply system, a boiler heated by a 60 kW resistor, a steam flow system and a complex condenser coupled with a heat exchanger. The feed water flow (sensor F3) is circulated via two feed pumps in parallel connection. Each pump is controlled by an on-off controller to maintain a constant water level (sensor L8) in the steam generator. The heat power (sensor Q4) depends on the pressure (sensor P7): when this pressure drops below a minimum value the heat resistance delivers maximum power and when the accumulator reaches a maximum pressure the electrical feed of the heat resistance is cut off. The expansion of the generated steam is realized by three valves in parallel connection. V4 is a manual bypass valve, simulating the pass around of the steam flow to the condenser. V5 is a controlled position valve. V6 is automatically controlled to maintain proper pressure to the condenser (sensor

P15). In an industrial plant, the steam flows to the turbine for generating power, but at the test stand, the steam is condensed and stored in a receiver tank and then returned to the steam generator.

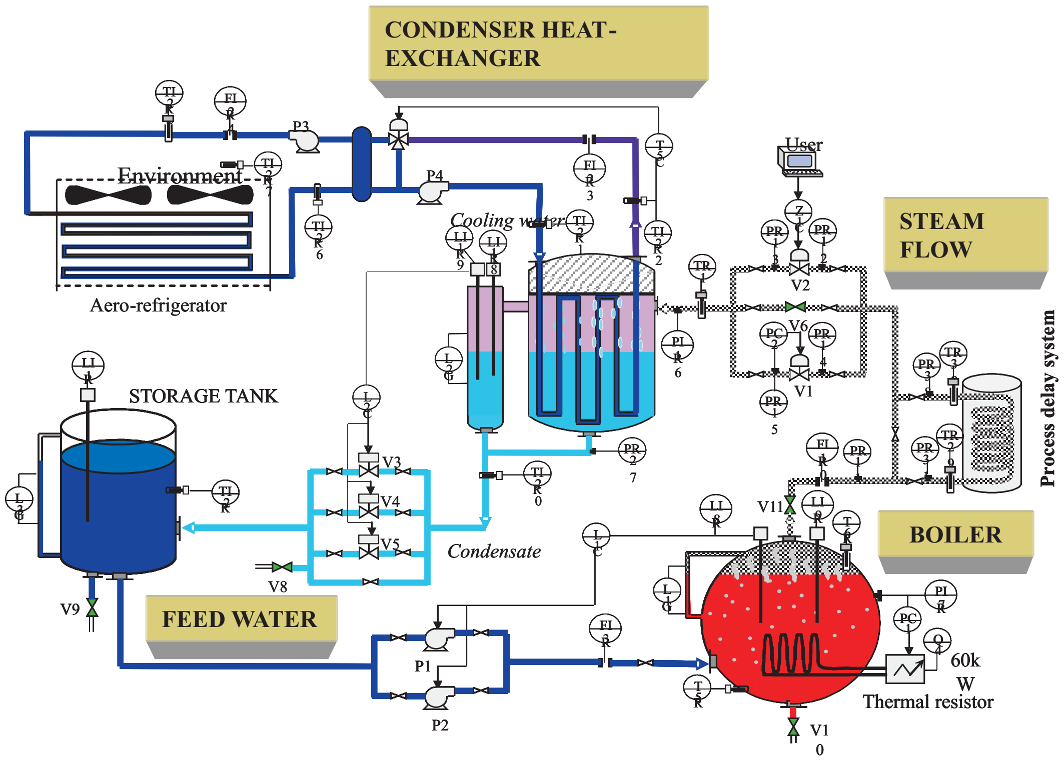


Figure 6.6: Flowsheet of the Steam Generator.

6.3.2.2 Testing scenario

The faulty scenario that has been considered consists of turning off the resistor $R1$ during a time period to simulate a shutdown. This affects the steam flow subsystem.

6.3.2.3 Process model

A model of the boiler is obtained through the next mass balance.

$$\frac{dm}{dt} = f_3 - f_{10} \tag{6.30}$$

where m is the mass inside the boiler, f_{10} is the mass output flow and f_3 is the mass input flow.

Its corresponding difference equation is

$$m(k) = m(k-1) + (f_3(k-1) - f_{10}(k-1)) * ts, \tag{6.31}$$

where t_s is the sampling time.

As the monitored variable m is not directly measurable, it is estimated (m_e) through the next equation.

$$m_e = v_{steam} * \rho_{steam} + v_{liquid} * \rho_{liquid} \tag{6.32}$$

where v_{liquid} is the volume of liquid (sensor L8), ρ_{liquid} is the liquid water density, which is a function of the pressure inside the boiler p_7 (sensor P7) and obtained using tables about the steam properties, v_{steam} is the volume of steam (given by $v_{steam} = v_{boiler} - v_{liquid}$) and ρ_{steam} is the steam density (also given by the steam tables).

Therefore, the analytical redundancy relation is

$$m(k) - m_e(k) = 0. \tag{6.33}$$

Table 6.1 shows the uncertainty associated to each measurement. The relative error e_r corresponds to the sensor precision and the absolute error e_a corresponds to the error introduced by the truncations of the used digital scale. In order to obtain the domain X' associated to a measurement x, the following formula is used.

$$X' = [x * (1 - e_r), x * (1 + e_r)] + [-e_a, e_a]. \tag{6.34}$$

6.3.2.4 Testing results

Figure 6.7 shows the main window of the online version of the fault detection tool for the tested scenario. In this case the sample time is 1 s and the used time window lengths are 10, 50, 100 and 200 s. Notice that in this version of the software, the measured output is plotted in yellow, the graph in the middle

Sensor	Relative error	Absolute error
f_3	1.6e-002	1.085e-004
f_{10}	0.01	0.0244
l_8	0.027	0.000073
p_7	0.005	0.0039

Table 6.1: Measurements uncertainty.

shows the faults and the one in the bottom indicates the window length. In this scenario, the fault begins at 240 s and ends at 360 s. The SQUALTRACK solver detects it from 353 s until 670 s.

Remark 6.3.1. *Notice that in this scenario there is a long period of false detections. It is due to the use of window length because, the faulty data is still used once the fault has disappeared.*
■

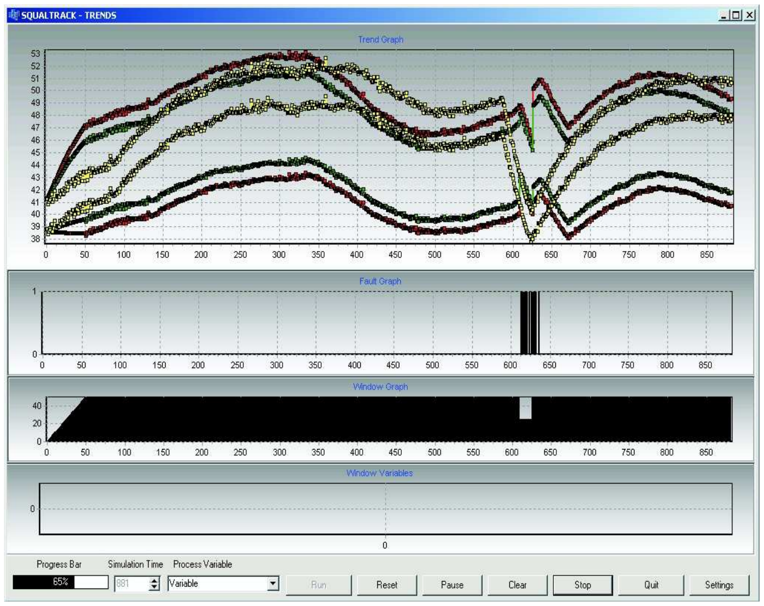


Figure 6.7: Boiler Leakage.

6.3.3 Fluid Catalytic Cracking plant

6.3.3.1 Process description

The FCC process consists of cracking heavy products in the presence of a catalyst. Heavy products come from atmospheric distillation and cracking allows lighter products with more value, mainly benzene, to be obtained. This process includes many devices: two regenerators, one reactor, one separation column, pipes, valves, etc. One difficulty in this process is the catalyst circulation loop. A global schematic of the process is shown in Figure 6.8.

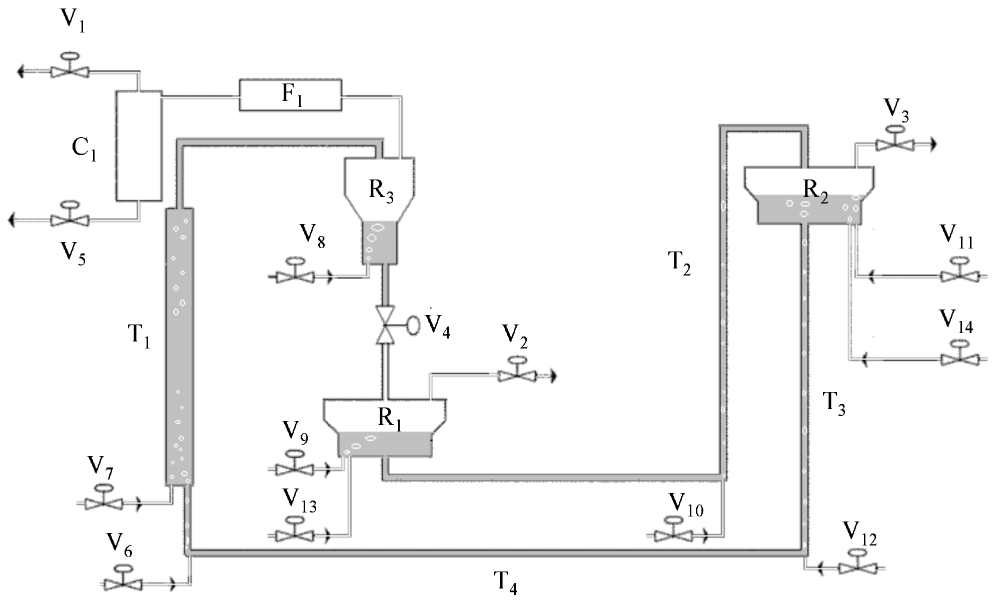


Figure 6.8: Schematic of FCC process.

6.3.3.2 Test scenario

The fault scenario that was considered was a fault in the first regenerator (R1) such that the air flow was abnormal.

6.3.3.3 Process model

In this case the model is a first-order transfer function obtained from a simplification of the FCC model. It models the dynamic relationship between the air flow entering to the regenerator R1 and its setpoint. The transfer function in the Laplace domain is

$$\frac{y(s)}{u(s)} = \frac{k}{1+\tau s}, \tag{6.35}$$

where k_s is the static gain and τ is the time constant.

The discrete-time model is then:

$$y_p(k) = \left(1 - \frac{t_s}{\tau}\right) y_m(k-1) + k_s \frac{T_s}{\tau} u_m(k-1), \tag{6.36}$$

where

- t_s is the sampling time,
- $y_p(k)$ is the output of the model at sample time k,
- $y_m(k-1)$ is the measurement of the output at time $k-1$ and
- $u_m(k-1)$ is the measurement of the input at time $k-1$.

The discrete-time equation for a window of length w is:

$$\begin{aligned} y_p(k|w) = \quad & \left(1 - \frac{t_s}{\tau}\right)^w y_m(k-w) + \ldots \\ & + \sum_{n=0}^{n=w-1} \left[\left(1 - \frac{t_s}{\tau}\right)^n k_s \frac{t_s}{\tau} u_m(k+(w-n-1)) \right]. \end{aligned} \tag{6.37}$$

The interval values of the parameters of the model to express their uncertainties have been obtained from process experts. They are:

- $\frac{t_s}{\tau} \in [0.97, 0.999]'$ and
- $K\frac{t_s}{\tau} \in [0.00095, 0.0315]'$.

Therefore, the analytical redundancy relation is

$$y_p(k|w) - y_m(k) = 0. \tag{6.38}$$

6.3.3.4 Test results

The test was performed using actual data from a scenario where the setpoint changed at time $t = 142$ s from 0 to $1,5$ and windows of lengths 2, 5, 10 and 15. Figure 6.9 shows that the fault was detected at different times from sample 285 to sample 353.

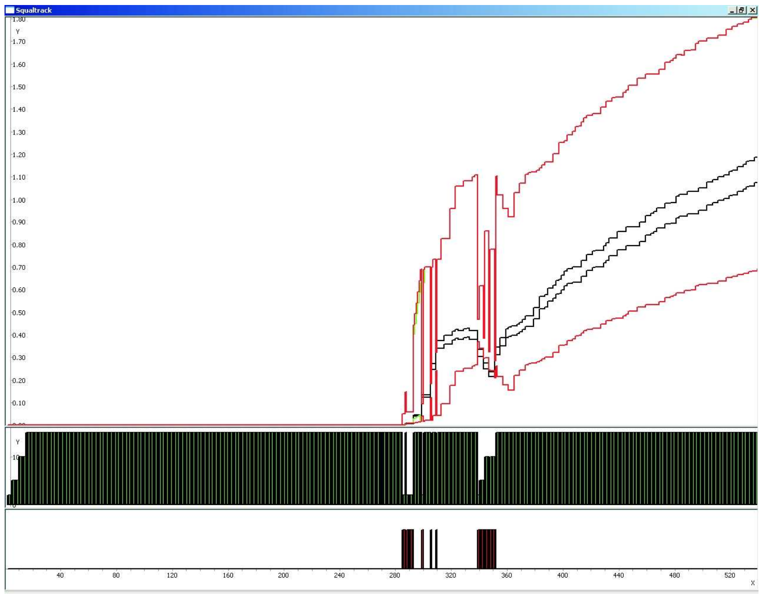


Figure 6.9: Faulty response of the air flow to a setpoint change.

Remark 6.3.2. *Details about the software implementation of the previous fault detection algorithm (SQUALTRACK solver) are explained in Chapter 8. The sources for introducing the previous examples to the SQUALTRACK solver are found in Appendix A.*
■

6.4 Conclusions

In this chapter, it has been shown how the problem of robust fault detection in dynamic processes can be stated as a problem of satisfying a quantified real constraint. The proposed technique is based on the principle of analytical redundancy and takes uncertainty into account by means of interval parameters and interval measurements. It also uses multiple time window lengths to maximize the detection of faults minimizing the computations. The modal interval inclusion test from Section 3.2.1.8, is used for the resolution of the resulting quantified real constraint. One of the contribution of the presented work consists of the implementation of a software tool called SQUALTRACK solver which has been applied to the detection of faults in different actual processes used in the European project CHEM. In the future, it is expected to extend this technique not to only detect but to diagnose faults. In this direction, some research has already been proposed by Calderón-Espinoza in Calderón-Espinoza *et al.* (2004).

Chapter 7

Application to Sailboat Control

This Chapter presents an application of the Quantified Set Inversion (QSI) algorithm explained in Chapter 4 to the sailboat control. More specifically, the QSI algorithm is used to find the *polar speed diagram* of a sailboat and then, feedback linearization techniques are applied to build a controller for the sailboat. Finally, simulation results are provided to demonstrate the viability of the method.

7.1 Introduction

In sailing, the sailor disposes of two actuators to control the speed and the orientation of a boat: the sail adjustment and the rudder adjustment. However, these actuators are not intuitive at all, specially when a precis control of the speed is desired, and requires a training period before acquiring a good handling of the boat. Automatically controlling the speed and the orientation of a sailboat has a practical interest. For example,

- During the mooring manoeuvre inside a harbor, where the speed of the sailboat has to be controlled not to surpass the speed limit.
- As a support system when crossing the oceans in solo (e.g. sleeping periods).
- For completely autonomous sailing (e.g. ecologic surveillance missions).

A proof of this interest are the two recent sailing competitions: the Microtransat Challenge Cup Microtransat (2006) held in Toulouse (France), which aims at building sailboats able to cross the Atlantic ocean in an autonomous way, and the Sailbot Sailbot (2006), another robotic sailboat competition held in Ontario (Canada).

Some previous works have already been proposed to tackle with the problem of the control of a sailboat Elkaim & Kelbley (2006); Elkaim *et al.* (2006). But, to our knowledge, this is the first approach which combines set computation techniques and feedback linearization for the control of a sailboat. This work has been partially presented in an international conference Herrero *et al.* (2005a) and a more complete paper has been submitted for publication in a journal Herrero *et al.* (2006).

7.2 Control strategy

In feedback linearization for control, the control output vector is usually foisted by the structure of the system to make the method working. This often implies that the control output vector does not coincide with the output desired by the user. In this section, a non-linear control schema based on Quantified Set Inversion (see Chapter 4) and feedback linearization, which allows to overcome this drawback in most of the cases, is proposed.

Consider the system described by the nonlinear differential equation:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{7.1}$$

where $\mathbf{x}$ and $\mathbf{u}$ are respectively the state vector and the input vector.

For some rather large condition on the system and for specific output vector $\mathbf{y} = \mathbf{g}(\mathbf{x})$, feedback linearization methods Fliess *et al.* (1995) make it possible to find a controller of the form

$$\mathbf{u} = \mathcal{R}_u(\mathbf{x}, \bar{\mathbf{y}}), \tag{7.2}$$

such that the output $\mathbf{y}$ converges to $\bar{\mathbf{y}}$. The vectors $\mathbf{y}$ and $\bar{\mathbf{y}}$ are called *control output vector* and the *desired control output vector*. Now, in many cases, the user wants to choose its own output $\mathbf{w} = \mathbf{h}(\mathbf{x})$ and not to have an output vector $\mathbf{y}$

foisted by the structure of the system to make the feedback linearization method working.

In this chapter, the problem of interest is to find a controller

$$\mathbf{u} = \mathcal{R}_w(\mathbf{x}, \bar{\mathbf{w}}) \tag{7.3}$$

such that the *user output vector* $\mathbf{w}$ converges to the *desired user output vector* $\bar{\mathbf{w}}$.

Define the set of all feasible user output vectors by

$$\Sigma_w = \{\mathbf{w} \in \mathbb{R}^m | (\exists \mathbf{x} \in \mathbb{R}^n)(\exists \mathbf{u} \in \mathbb{R}^m)(\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{0} \wedge \mathbf{w} = \mathbf{h}(\mathbf{x}))\}. \tag{7.4}$$

Since Σ_w is defined by $n + m$ equations for $m + 2n$ variables, except for atypical situations, the set Σ_w has a nonempty volume.

Firstly, the QSI algorithm presented in Chapter 4 is used to characterize the inner and an outer approximations of Σ_w. Then, the user will be allowed to choose any point $\bar{\mathbf{w}}$ inside Σ_w. From $\bar{\mathbf{w}}$, we will then compute some corresponding $\bar{\mathbf{x}}$ and $\bar{\mathbf{u}}$ such that $\mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = \mathbf{0}, \bar{\mathbf{w}} = \mathbf{h}(\bar{\mathbf{x}})$. Note the solution pair $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ may be not unique. From $\bar{\mathbf{x}}$, we shall compute $\bar{\mathbf{y}} = \mathbf{g}(\bar{\mathbf{x}})$. Then the controller $\mathcal{R}_u(\mathbf{x}, \bar{\mathbf{y}})$ will compute $\mathbf{u}$ such that $\mathbf{y}$ converges to $\bar{\mathbf{y}}$. As a consequence, $\mathbf{x}$ will tend to $\bar{\mathbf{x}}$ and $\mathbf{w}$ to $\bar{\mathbf{w}}$. Figure 7.1 graphically represents the proposed control schema.

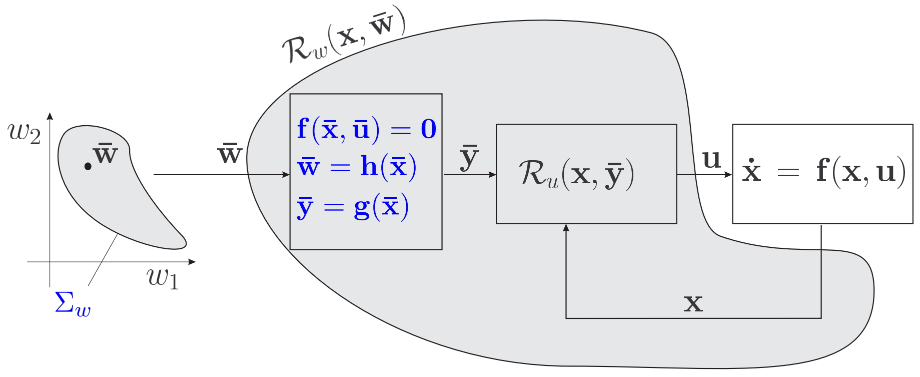

$\mathcal{R}_w(\mathbf{x}, \bar{\mathbf{w}})$
w_2
$\bar{\mathbf{w}}$
w_1
Σ_w
$\bar{\mathbf{w}}$
$\mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = \mathbf{0}$
$\bar{\mathbf{w}} = \mathbf{h}(\bar{\mathbf{x}})$
$\bar{\mathbf{y}} = \mathbf{g}(\bar{\mathbf{x}})$
$\bar{\mathbf{y}}$
$\mathcal{R}_u(\mathbf{x}, \bar{\mathbf{y}})$
$\mathbf{u}$
$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$
$\mathbf{x}$

Figure 7.1: Control diagram.

7.3 Sailboat control application

The proposed control scheme can be divided into two parts: A first part, which is executed off-line and explained in Section 7.5, and consists of characterizing an inner and an outer approximation of the set of admissible set-points which can be chosen by the user. This set corresponds to a well known sailing diagram called *polar diagram*, which is the set of all pairs (v, θ) that can be reached by the boat, in a cruising behavior. For this purpose, the Quantified Set Inversion (QSI) algorithm, explained in Chapter 4, is used. A second part, which is executed online and explained in Section 7.6, corresponds to a particular feedback linearization controller.

However, the designed feedback linearization controller does not accept (v, θ) as linearizing control output vector due to the amount of singularities that generates the linearization procedure. For this reason, the pre-compensator module, explained in Section 7.7, is included to transform the set-point chosen by the user $(\bar{v}, \bar{\theta})$ into a suitable control output vector for the controller, which corresponds to the sail adjustment and the orientation of the boat $(\bar{\delta}_s, \bar{\theta})$. Finally, the controller will compute the necessary sail and rudder adjustments $(\bar{\delta}_s, \bar{\delta}_r)$ such that (v, θ, δ_s) tends to $(\bar{v}, \bar{\theta}, \bar{\delta}_s)$. Figure 7.2 graphically shows the proposed control scheme.

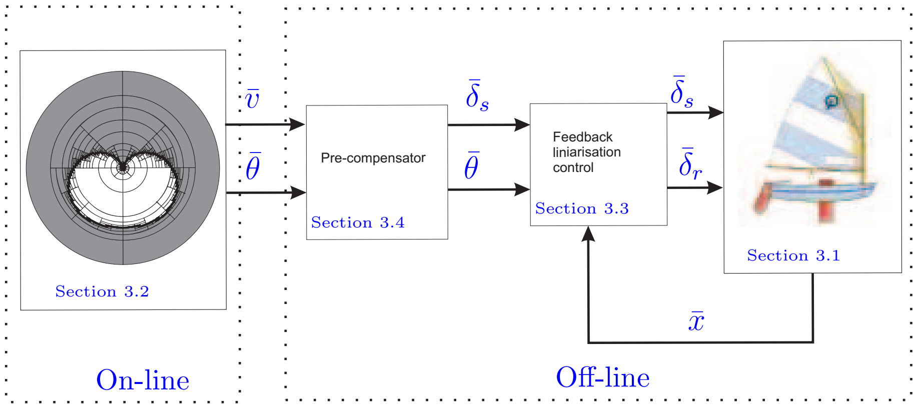


Figure 7.2: Sailboat control scheme.

7.4 Sailboat model

The sailboat represented on Figure 7.3 (taken from Jaulin (2001b)) is described by the following state equations

$$\left\{\begin{array}{lcll}
\dot{x} & = & v\cos\theta, & \text{(i)}\\
\dot{y} & = & v\sin\theta - \beta v_w, & \text{(ii)}\\
\dot{\theta} & = & \omega, & \text{(iii)}\\
\dot{\delta}_s & = & u_1, & \text{(iv)}\\
\dot{\delta}_r & = & u_2, & \text{(v)}\\
\dot{v} & = & \frac{f_s\sin\delta_s - f_r\sin\delta_r - \alpha_f v}{m}, & \text{(vi)}\\
\dot{\omega} & = & \frac{(\ell - r_s\cos\delta_s)f_s - r_r\cos\delta_r f_r - \alpha_\theta\omega}{J}, & \text{(vii)}\\
f_s & = & \alpha_s\left(v_w\cos\left(\theta+\delta_s\right) - v\sin\delta_s\right), & \text{(viii)}\\
f_r & = & \alpha_r v\sin\delta_r, & \text{(ix)}
\end{array}\right. \tag{7.5}$$

where $\dot{x}, \dot{y}, \ldots$ represents the derivatives of $x, y, \ldots$ with respect to the time t. The state vector $\mathbf{x} = (x, y, \theta, \delta_s, \delta_r, v, \omega)^{\mathrm{T}} \in \mathbb{R}^7$ is composed by

- the coordinates x, y of the inertial center G of the boat
- the orientation θ,
- the sail angle δ_s
- the rudder angle δ_r
- the tangential speed v of G
- the angular velocity ω of the boat around G.

The intermediate variables are

- the thrust force f_s of the wind on the sail,
- the force f_r of the water on the rudder.

The parameters (that are assumed to be known) are

- the speed v_w of the wind,
- the distance r_r between the rudder and G,

- the distance r_s between the mast and G,
- the rudder lift α_r,
- the sail lift α_s,
- the tangential friction α_f of the boat with respect to the water,
- the angular friction α_θ of the boat with respect to the water,
- the angular inertia J of the boat,
- the distance ℓ between the mast and the thrust center of the sail,
- and the drift coefficient β.

The parameters values are chosen as

$$\begin{aligned} &\beta = 0.05, r_s = 1, r_r = 2, \ell = 1, v_w = 10, \\ &m = 1000, J = 2000, \alpha_f = 60, \\ &\alpha_\theta \in 500, \alpha_s = 500, \alpha_r = 300. \end{aligned} \tag{7.6}$$

The inputs u_1 and u_2 of the system are the derivatives of the angles δ_s and δ_r.

7.5 Polar diagram of a sailboat

The *polar diagram* of the sailboat is defined by the set $\mathbb{S}$ of all pairs (θ, v) that can be potentially reached by the boat, in a cruising behavior.

During a cruising behavior of the boat, the speed of the boat, its course, its angular velocity, ... are supposed to be constant, i.e.,

$$\dot{\theta} = 0, \dot{\delta}_s = 0, \dot{\delta}_r = 0, \dot{v} = 0, \dot{\omega} = 0. \tag{7.7}$$

From Equation 7.5, we get

$$\left\{ \begin{array}{lll} 0 & = & \frac{f_s \sin\delta_s - f_r \sin\delta_r - \alpha_f v}{m}, \\ 0 & = & \frac{(\ell - r_s \cos\delta_s) f_s - r_r \cos\delta_r f_r}{J}, \\ f_s & = & \alpha_s \left(v_w \cos(\theta + \delta_s) - v \sin\delta_s \right), \\ f_r & = & \alpha_r v \sin\delta_r. \end{array} \right. \tag{7.8}$$

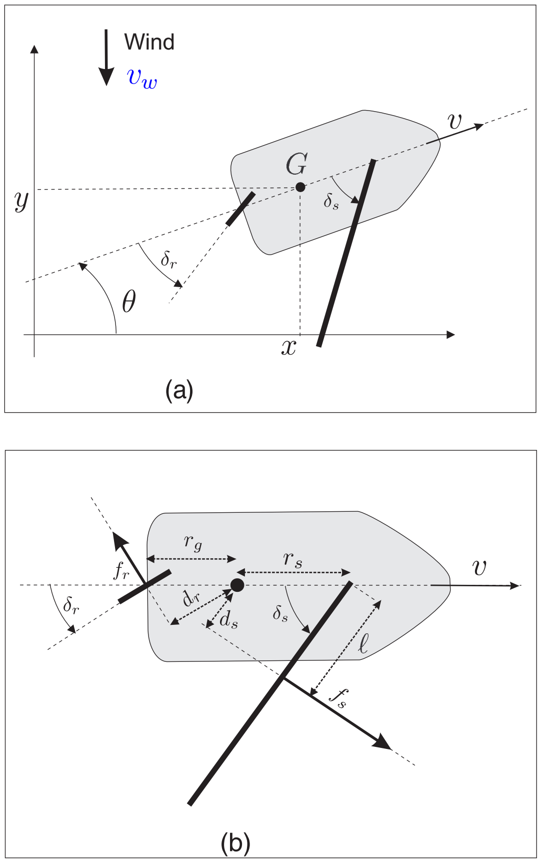


Figure 7.3: Sailboat model.

which is equivalent to

$$\left\{\begin{array}{cl} \alpha_s\left(v_w\cos\left(\theta+\delta_s\right)-v\sin\delta_s\right)\sin\delta_s- & \\ \alpha_r v\sin^2\delta_r-\alpha_f v & = \ 0, \\ \left(\ell-r_s\cos\delta_s\right)\alpha_s\left(v_w\cos\left(\theta+\delta_s\right)-v\sin\delta_s\right)- & \\ r_r\alpha_r v\sin\delta_r\cos\delta_r & = \ 0. \end{array}\right. \quad (7.9)$$

The polar diagram is the set Σ of all feasible vectors (v,θ) in a cruising regime, i.e.,

$$\Sigma=\left\{(v,\theta)\mid\exists\delta_r,\exists\delta_s,\mathbf{f}\left(v,\theta,\delta_r,\delta_s\right)=\mathbf{0}\right\}, \quad (7.10)$$

where

$$\mathbf{f}(v,\theta,\delta_r,\delta_s) = \begin{pmatrix} \alpha_s\left(v_w\cos(\theta+\delta_s)-v\sin\delta_s\right)\sin\delta_s - \\ \alpha_r v\sin^2\delta_r - \alpha_f v \\ (\ell - r_s\cos\delta_s)\,\alpha_s\left(v_w\cos(\theta+\delta_s)-v\sin\delta_s\right) - \\ r_r\alpha_r v\sin\delta_r\cos\delta_r \end{pmatrix}. \tag{7.11}$$

7.5.1 Transformation of the problem

Characterizing an inner approximation for the set Σ given by Equation 7.10 cannot be done using the QSI algorithm. The reason for that is that the dimension n_f of $\mathbf{f}$ is equal to 2. In this section, we shall eliminate the variable δ_r using symbolic calculus to cast into the case where $n_f = 1$.

Since

$$\sin^2\delta_r = \frac{1-\cos(2\delta_r)}{2} \text{ and } \sin\delta_r\cos\delta_r = \frac{\sin(2\delta_r)}{2}, \tag{7.12}$$

and since $\sin^2(2\delta_r)+\cos^2(2\delta_r)-1=0$, we get that

$$\begin{aligned} &\left(-1+\tfrac{2}{\alpha_r v}\left(\alpha_s\left(v_w\cos(\theta+\delta_s)-v\sin\delta_s\right)\sin\delta_s-\alpha_f v\right)\right)^2 \\ &+\left(\tfrac{2\alpha_s}{r_r\alpha_r v}\left(\ell-r_s\cos\delta_s\right)\left(v_w\cos(\theta+\delta_s)-v\sin\delta_s\right)\right)^2 \\ &-1=0, \end{aligned} \tag{7.13}$$

i.e.,

$$\begin{aligned} &\left((\alpha_r+2\alpha_f)\,v-2\alpha_s v_w\cos(\theta+\delta_s)\sin\delta_s+2\alpha_s v\sin^2\delta_s\right)^2 \\ &+\left(\tfrac{2\alpha_s}{r_r}\left(\ell-r_s\cos\delta_s\right)\left(v_w\cos(\theta+\delta_s)-v\sin\delta_s\right)\right)^2 \\ &-\alpha_r^2 v^2=0. \end{aligned} \tag{7.14}$$

The polar diagram can thus be written as

$$\Sigma = \left\{(\theta,v)\middle|(\exists\delta_s\in[-\frac{\pi}{2},\frac{\pi}{2}])\mid f_1(\theta,v,\delta_s)=0\right\}, \tag{7.15}$$

where $f_1(\theta,v,\delta_s)$ is given by

$$\begin{aligned} &f_1(\theta,v,\delta_s)= \\ &\left((\alpha_r+2\alpha_f)\,v-2\alpha_s v_w\cos(\theta+\delta_s)\sin\delta_s+2\alpha_s v\sin^2\delta_s\right)^2 \\ &+\left(\tfrac{2\alpha_s}{r_r}\left(\ell-r_s\cos\delta_s\right)\left(v_w\cos(\theta+\delta_s)-v\sin\delta_s\right)\right)^2-\alpha_r^2 v^2. \end{aligned} \tag{7.16}$$

Now, the problem stated by Equation 7.15 can be solved by QSI algorithm. Notice that other existing solvers, like Dao (2005); Ratschan (2002a, 2005), can not find an inner approximation of Equation 7.15 due to the presence of the equality predicate.

7.5.2 Resolution

By using the QSI algorithm with a precision of $\epsilon = 0.02$, in less than 60 seconds on a Pentium IV M 1.5 GHz, the result expressed in polar coordinates and showed in Figure 7.4 is obtained, where the white area corresponds to the set of points (θ, v) which can be potentially reached by the sailboat, the grey area corresponds to the set of non feasible points and the black area is undefined.

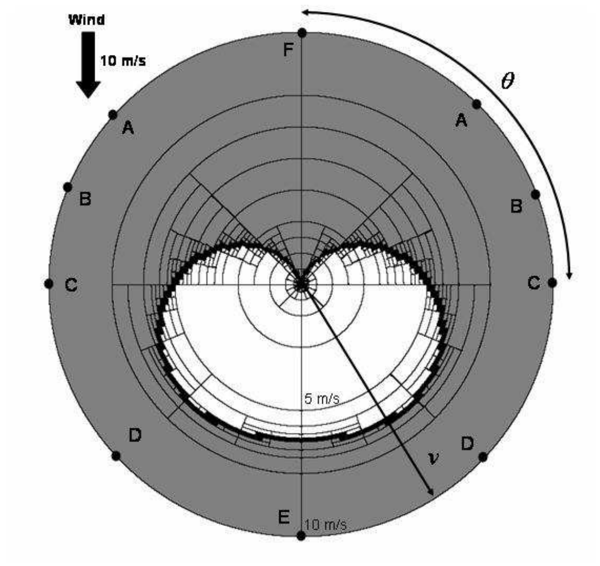


Figure 7.4: Polar diagram obtained with the QSI solver.

7.6 Feedback linearization control

To apply the feedback linearization method it is necessary to differentiate the state variables, one or more times with respect to time till the inputs u_1 or u_2

appear.

In Equations 7.5, only $\dot{\delta}_s$ and $\dot{\delta}_g$ are algebraically related to $\mathbf{u}$, the others are related to $\mathbf{u}$, but indirectly ,i.e., differentially. It is only necessary to differentiate these equations which are no related to $\mathbf{u}$, i.e., $\dot{x}$, $\dot{y}$, $\dot{\theta}$, $\dot{v}$ and $\dot{\omega}$. We get

$$\left\{\begin{array}{lll} \ddot{x} & = & \dot{v}\cos\theta - v\dot{\theta}\sin\theta, \\ \ddot{y} & = & \dot{v}\sin\theta + v\dot{\theta}\cos\theta, \\ \ddot{\theta} & = & \dot{\omega}, \\ \ddot{v} & = & \frac{\dot{f}_s\sin\delta_s + f_s u_1\cos\delta_s - \dot{f}_r\sin\delta_r - f_r u_2\cos\delta_r - \alpha_f\dot{v}}{m}, \\ \ddot{\omega} & = & \frac{u_1 r_s\sin\delta_s f_s + (\ell - r_s\cos\delta_s)\dot{f}_s}{J} \\ & + & \frac{r_r\left(u_2\sin\delta_r f_r - \cos\delta_r\dot{f}_r\right) - \alpha_\theta\dot{\omega}}{J}. \end{array}\right. \tag{7.17}$$

with

$$\left\{\begin{array}{lll} \dot{f}_s & = & -\alpha_s v_w\left(\omega + u_1\right)\sin\left(\theta + \delta_s\right) - \alpha_s\dot{v}\sin\delta_s - \\ & & \alpha_s v u_1\cos\delta_s, \\ \dot{f}_r & = & \alpha_r\left(\dot{v}\sin\delta_r + v u_2\cos\delta_r\right). \end{array}\right. \tag{7.18}$$

Notice that, as $\dot{v}, \dot{\theta}, \dot{\omega}$ are analytic functions of the state (see Equation 7.5), it is possible to consider that we have an analytic function of $\ddot{x}, \ddot{y}, \ddot{\theta}, \ddot{v}, \ddot{\omega}$ depending on the state and inputs. It is necessary to differentiate again all these quantities which do not algebraically depend on $\mathbf{u}$, which means $\ddot{x}$, $\ddot{y}$ et $\ddot{\theta}$. Then we get

$$\left\{\begin{array}{lll} \dddot{x} & = & \ddot{v}\cos\theta - 2\dot{v}\dot{\theta}\sin\theta - v\ddot{\theta}\sin\theta - v\dot{\theta}^2\cos\theta, \\ \dddot{y} & = & \ddot{v}\sin\theta + 2\dot{v}\dot{\theta}\cos\theta + v\ddot{\theta}\cos\theta - v\dot{\theta}^2\sin\theta, \\ \dddot{\theta} & = & \ddot{\omega}. \end{array}\right. \tag{7.19}$$

As all these quantities algebraically dependent on $\mathbf{u}$, it is not necessary to differentiate more. Notice again, that the Equations 7.19 can be interpreted (via Equations 7.5 and 7.17) as analytic expressions of $\dddot{x}$, $\dddot{y}$, $\dddot{\theta}$ depending on de state and inputs.

As the system has two inputs, it is necessary to choose two outputs in order to do the feedback linearization. A first possibility consists on choosing as outputs, the speed $y_1 = v$ and the orientation $y_2 = \theta$. This choice can be justified because $\mathbf{y}$ is a flat output for the sub-system described by the (i-vii) relations of Equation 7.5 (for more information about about flat outputs see Fliess *et al.* (1995)). A brief demonstration can be done by the algorithm of the dynamic extension Fliess *et al.*

(1995). This demonstration is a direct consequence of the next calculus which tries to determine the linearizing loop. We have

$$\begin{aligned}\begin{pmatrix}\ddot{y}_1\\ \dddot{y}_2\end{pmatrix} &= \begin{pmatrix}\ddot{v}\\ \dddot{\theta}\end{pmatrix}\\ &= \mathbf{A}_1(\mathbf{x})\begin{pmatrix}u_1\\ u_2\end{pmatrix} + \mathbf{A}_2(\mathbf{x})\begin{pmatrix}\dot{f}_s\\ \dot{f}_r\end{pmatrix} + \mathbf{b}_1(\mathbf{x}),\end{aligned} \tag{7.20}$$

with

$$\begin{aligned}\mathbf{A}_1(\mathbf{x}) &= \begin{pmatrix}\frac{1}{m}f_s\cos\delta_s & -\frac{1}{m}f_r\cos\delta_r\\ \frac{r_s}{J}f_s\sin\delta_s & \frac{r_r}{J}f_r\sin\delta_r\end{pmatrix},\\ \mathbf{A}_2(\mathbf{x}) &= \begin{pmatrix}\frac{1}{m}\sin\delta_s & -\frac{1}{m}\sin\delta_r\\ \frac{\ell}{J}-\frac{r_s}{J}\cos\delta_s & -\frac{r_r}{J}\cos\delta_r\end{pmatrix},\\ \mathbf{b}_1(\mathbf{x}) &= \begin{pmatrix}-\frac{\alpha_f}{m}\dot{v}\\ -\frac{\alpha_\theta}{J}\dot{\omega}\end{pmatrix}.\end{aligned} \tag{7.21}$$

and

$$\begin{pmatrix}\dot{f}_s\\ \dot{f}_r\end{pmatrix} = \mathbf{A}_3(\mathbf{x})\begin{pmatrix}u_1\\ u_2\end{pmatrix} + \mathbf{b}_2(\mathbf{x}), \tag{7.22}$$

with

$$\mathbf{A}_3(\mathbf{x}) = \begin{pmatrix}-\alpha_s\left(v_w\sin\left(\theta+\delta_s\right)+v\cos\delta_s\right) & 0\\ 0 & \alpha_r v\cos\delta_r\end{pmatrix}, \tag{7.23}$$

$$\mathbf{b}_2(\mathbf{x}) = \begin{pmatrix}-\alpha_s\left(v_w\omega\sin\left(\theta+\delta_s\right)+\dot{v}\sin\delta_s\right)\\ \alpha_r\dot{v}\sin\delta_r\end{pmatrix}. \tag{7.24}$$

Which leads to a relation of the form

$$\begin{aligned}\begin{pmatrix}\ddot{y}_1\\ \dddot{y}_2\end{pmatrix} &= \mathbf{A}_1\mathbf{u} + \mathbf{A}_2\left(\mathbf{A}_3\mathbf{u}+\mathbf{b}_2\right)+\mathbf{b}_1\\ &= \left(\mathbf{A}_1{+}\mathbf{A}_2\mathbf{A}_3\right)\mathbf{u} + \mathbf{A}_2\mathbf{b}_2 + \mathbf{b}_1\\ &= \mathbf{A}(\mathbf{x})\mathbf{u} + \mathbf{b}(\mathbf{x}).\end{aligned} \tag{7.25}$$

For imposing the quantities $(\ddot{y}_1, \dddot{y}_2)$ to a given set-point

$$\mathbf{v} \;=\; \left(v_1, v_2\right)^{\mathrm{T}}, \tag{7.26}$$

it is necessary to take as inputs

$$\mathbf{u} = \mathbf{A}^{-1}(\mathbf{x})\left(\mathbf{v}-\mathbf{b}(\mathbf{x})\right). \tag{7.27}$$

The equations of the looped system are

$$\left\{ \begin{array}{lll} \ddot{y}_1 & = & v_1, \\ \dddot{y}_2 & = & v_2. \end{array} \right. \tag{7.28}$$

The looped system has dimension 5 instead of dimension 7 for the initial system. The loss of control over two state variables: x an y, has been produced. The singularities of the resulting linearizing control are solutions of the equation

$$\det \mathbf{A}\left(\mathbf{x}\right) = 0, \tag{7.29}$$

which has many solutions. For this reason, we have decided to choose another output which generates less singularities. Let us choose now as outputs the sail adjustment $y_1 = \delta_s$ and the orientation $y_2 = \theta$. We have

$$\begin{aligned} \begin{pmatrix} \dot{y}_1 \\ \dddot{y}_2 \end{pmatrix} &= \begin{pmatrix} \dot{\delta}_s \\ \dddot{\theta} \end{pmatrix} \\ &= \mathbf{A}_1(\mathbf{x}) \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} + \mathbf{A}_2(\mathbf{x}) \begin{pmatrix} \dot{f}_s \\ \dot{f}_r \end{pmatrix} + \mathbf{b}_1(\mathbf{x}), \end{aligned} \tag{7.30}$$

with

$$\begin{aligned} \mathbf{A}_1(\mathbf{x}) &= \begin{pmatrix} 1 & 0 \\ \frac{r_s f_s \sin\delta_s}{J} & \frac{r_r f_r \sin\delta_r}{J} \end{pmatrix}, \\ \mathbf{A}_2(\mathbf{x}) &= \begin{pmatrix} 0 & 0 \\ \frac{\ell - r_s \cos\delta_s}{J} & -\frac{r_r \cos\delta_r}{J} \end{pmatrix}, \\ \mathbf{b}_1(\mathbf{x}) &= \begin{pmatrix} 0 \\ -\frac{\alpha_\theta \dot{\omega}}{J} \end{pmatrix}, \end{aligned} \tag{7.31}$$

where $\dot{f}_s$ et $\dot{f}_r$ are given by the Equation 7.22. Then, we have a relation of the form

$$\begin{aligned} \begin{pmatrix} \dot{y}_1 \\ \dddot{y}_2 \end{pmatrix} &= \mathbf{A}_1\mathbf{u} + \mathbf{A}_2\left(\mathbf{A}_3\mathbf{u} + \mathbf{b}_2\right) + \mathbf{b}_1, \\ &= \left(\mathbf{A}_1 + \mathbf{A}_2\mathbf{A}_3\right)\mathbf{u} + \mathbf{A}_2\mathbf{b}_2 + \mathbf{b}_1, \\ &= \mathbf{A}(\mathbf{x})\mathbf{u} + \mathbf{b}(\mathbf{x}). \end{aligned} \tag{7.32}$$

In order to impose $(\dot{y}_1, \dddot{y}_2)$ to a given set-point $\mathbf{v} = (v_1, v_2)$, it is necessary to take

$$\mathbf{u} = \mathbf{A}^{-1}(\mathbf{x})\left(\mathbf{v} - \mathbf{b}(\mathbf{x})\right). \tag{7.33}$$

The looped system is governed by the differential equations

$$\begin{cases} \dot{y}_1 & = & v_1, \\ \dddot{y}_2 & = & v_2. \end{cases} \tag{7.34}$$

which are linear and decoupled. The linear system is of 4th order instead of 7th. We have lost the control over 3 variables which are x, y and v. The loss of control over x and y was predictable (it is desired that the sailboat advances and it is natural that it corresponds to an instability for the variables x and y). Referring to the loss of control over v, it does not have any consequence because the associated zero dynamic is stable. How can be conceived that the sailboat could maintain a sail and a rudder adjustments avoiding the speed to converge to an infinity value?

Let us determine the singularities of the linearizing loop. We can easily prove that

$$\det\left(\mathbf{A}(\mathbf{x})\right) = \frac{r_r}{J}\left(f_r \sin\delta_r - v\alpha_r\cos^2\delta_r\right), \tag{7.35}$$

is null if

$$v\left(2\sin^2\delta_r - 1\right) = 0, \tag{7.36}$$

i.e. if

$$v = 0 \text{ or } \delta_r = \frac{\pi}{4} + k\frac{\pi}{2}. \tag{7.37}$$

Such configuration corresponds to a singularity which should be avoided.

7.7 Pre-compensator module

As shown in Section 7.6, it has not been possible to choose the speed v and the orientation θ of the sailboat as linearizing outputs for the feedback linearization controller because of the amount of singularities that are provoked. Instead of that, the sail adjustment δ_s and the orientation of the sailboat θ have been chosen. As we want to control v and θ, it is necessary to introduce a pre-compensator which allows to transform the set-point chosen by the user ($\bar{v}$,$\bar{\theta}$) to an admissible set-point by the controller ($\bar{\delta_s}$,$\bar{\theta}$).

Fixed $(\bar{v},\bar{\theta})$ chosen by the user from the polar diagram, a numerically validated dichotomy algorithm, based on branch-and-bound techniques and Modal Interval Analysis, is applied for finding $\bar{\delta_s}$. The algorithm proceeds as follows; Given an initial range for the sail adjustment Δ_s, it is bisected into two subintervals $\Delta_s^{(1)}$ and $\Delta_s^{(2)}$, then, the next existence test is carried out over the resulting intervals

$$(\exists \delta_s \in \Delta'_s) f_1(\bar{\theta}, \bar{v}, \delta_s) = 0, \tag{7.38}$$

where $f_1(\theta, v, \delta_s)$ is expressed by Equation 7.16.

This test is easily done using Modal Interval Analysis, which transforms the logical statement of Equation 7.38 into an interval inclusion test by means of the *modal interval inclusion test* presented in Section 3.2.1.8. If the test is not verified for one of the subintervals it is eliminated, if both subintervals are eliminated, it means that no solution exists. Finally, if the test is positive for both subintervals, one of them is arbitrarily chosen and the other subinterval is rejected. Notice that multiple solutions could exist and the one given by our algorithm is not necessarily the optimal solution in terms of sailing. This bisection procedure is recursively repeated till a small enough interval is achieved for δ_s. As the controller requires a punctual value as input, the center of the resulting interval is chosen. Algorithm 7 shows the proposed algorithm and Figure 7.5 shows a graphical example of the same algorithm.

Algorithm 7 Existence Algorithm

Input: $f_1(\bar{\theta}, \bar{v}, \delta_s), \Delta'_s$ and ϵ.

Output: $\bar{\delta}_s$.

```
 1: Enqueue $\Delta'_s$ to $List$;
 2: while $List$ not empty do
 3:     Dequeue $\Delta'_s$ from $List$;
 4:     if $(\exists \delta_s \in \Delta'_s) f_1(\bar{\theta}, \bar{v}, \delta_s) = 0$ then
 5:         if $Width(\Delta'_s) < \epsilon$ then
 6:             return Center($\Delta'_s$);
 7:         else
 8:             Bisect $\Delta'_s$ and enqueue the resulting intervals to $List$;
 9:         end if
10:     end if
11: end while
```

- *List*: List of intervals.
- Enqueue: The result of adding an interval to *List*.
- Dequeue: The result of extracting an interval from *List*.
- $Width(\Delta'_s)$: Function returning the size of the interval Δ'_s,
- $Center(\Delta'_s)$: Function returning the center of the interval Δ'_s,
- ϵ: A real value representing the desired precision.

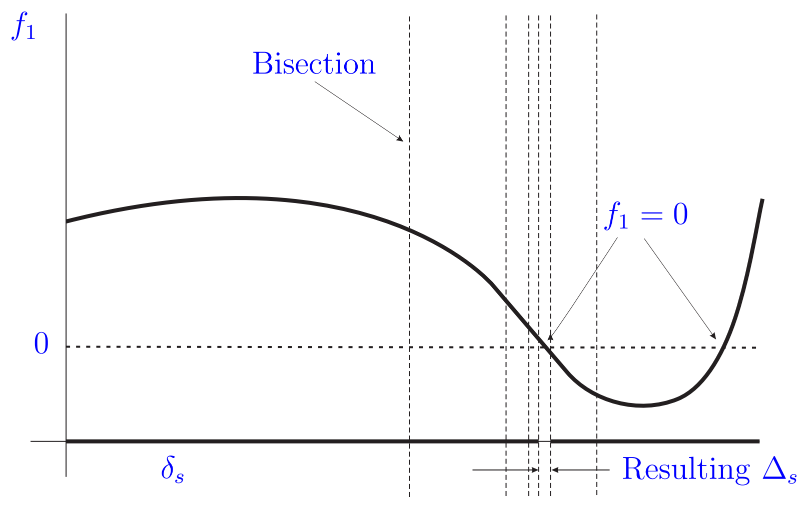


Figure 7.5: Existence algorithm.

7.8 Simulation results

The presented control schema has been informatically implemented in order to show its viability. An available demonstration software is available at the following URL Herrero & Jaulin (2006). Its use consists of selecting from the polar diagram the sequence of desired set-points by clicking with the mouse over it. Initial conditions like position x, y, orientation θ and speed v can also be chosen. Figure 7.6 shows a simulation sequence of the required manoeuver to moor the sailboat inside a harbor. Notice that the transitory states between two set-points have been neglected. However, this simplification does not affect the simulation results.

7.9 Conclusions

A frequent drawback in feedback linearization control is the amount of singularities on the resulting controller. For this reason, sometimes the choice of the control output is foisted by the control law and not by the user requirements. An original control schema based on Quantified Set Inversion techniques and

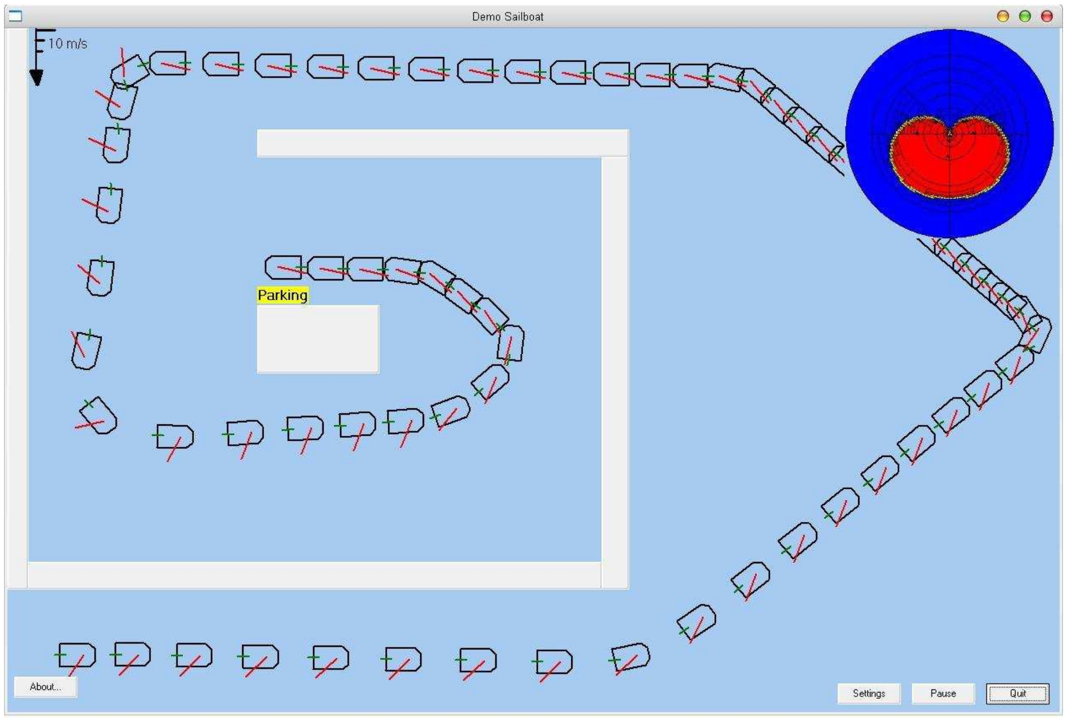


Figure 7.6: Manoeuver to moor the sailboat inside a harbor.

feedback linearization has been presented to overcome this drawback in most of the cases. A sailing boat control application has been solved to illustrate the methodology.

Chapter 8

Implementation

This chapter describes in a high-level way the implementation work, carried out by the author of this thesis, of the different algorithms which have been presented in the previous chapters. The idea of this chapter is not to enter into low level implementation details but to show the architecture of the developed software and to give an idea about the programming effort which has been carried out.

8.1 Implementation philosophy

The philosophy which has been followed for the implementation of the set of algorithms developed in this thesis is based on the following principles:

- **Reliability**: All the implemented algorithms are intended to be numerically guaranteed, which means that they must not provide false results due to numerical errors.
- **Maintainability**: The sources are intended to be as clear as possible in order to facilitate its understanding and to allow easy modifications.
- **Performance**: The implemented algorithms have a high computational complexity, then, efficient executable binaries are desired.
- **Portability**: The sources are desired to be as portable as possible between different operating systems and platforms.

- **Open source**: All the used programming tools are intended to be freely distributed and open source.
- **Easy use**: The resulting solvers have to be easy to use for the final user.

These principles have conditioned the selection of the programming tools, which are briefly described in the next section.

8.2 Programming tools

This section describes the set of programming tools which have been used for the implementation of the algorithms presented in the previous chapters.

8.2.1 C++ language

C++ Eckel (2000) is an *object oriented programming* language which implements *data abstraction* using a concept called *classes*, along with other features to allow object-oriented programming (OOP). Parts of the C++ program are easily reusable and extensible without needing big modifications of the source. C++ adds a concept called *operator overloading* not seen in the earlier OOP languages and it makes the creation of libraries much cleaner.

C++ maintains aspects of the C programming language, yet has features which simplify memory management. Additionally, some of the features of C++ allow low-level access to memory but also contain high level features.

Therefore, C++ programming language offers an interesting ratio between the different implementation principles presented in Section 8.1.

8.2.1.1 Code::Blocks and GCC compiler

Code::Blocks Code::Blocks (2006) is Interface Development Environment (IDE) which has been chosen for the implementation of the algorithms presented in the previous chapters. Code::Blocks is an open source, cross platform free C++ IDE, which was designed, right from the start, to be extensible and configurable. Code::Blocks supports multiple compiler like: GCC (MingW / Linux GCC), MSVC++, Digital Mars, Borland C++ 5.5 and Open Watcom. From the supported compilers, the GCC compiler GCC (2006) has been selected because it strictly follows the ANSI C++ standard. GCC is the compiler system of the GNU environment (GNU is a recursive acronym for ”GNU’s Not UNIX”) GNU (2006). GNU is a UNIX-compatible operating system, being developed by the *Free Software Foundation* FSF (2006), and distributed under the GNU Public License (GPL).

8.2.2 Standard Template Library

Many of the developed algorithm in this thesis are based on branch-and-bound techniques. For their implementation, the use of container classes like lists and vectors are intensely used. With the purpose of facilitating the programming task and obtaining a clear, robust and portable code, the so-called Standard Template Library (STL) STL (2006) has been employed.

The STL is a collection of container classes (e.g. list, vector,...), generic algorithms and related components that can greatly simplify many programming tasks in C++. STL comes with most of the existing C++ compilers (e.g. MS Visual C++, Borland Builder, GCC).

8.2.3 Spirit Parser framework

In order to provide solvers with friendly user interfaces, which allow to introduce the problems in an easy way, the use of a *parser* has been considered suitable.

In computer science, parsing is the process of analyzing an input sequence (read from a file or a keyboard, for example) in order to determine its grammatical structure with respect to a given formal grammar. It is formally named syntax analysis. A parser is a computer program that carries out this task. A parser can be implemented with native C++ language, however, there exist already developed frameworks which enormously facilitate this task (e.g. Lex/Yacc LEX-YACC (2006), Spirit de Guzman (2006)).

From the set of existing parser frameworks, the Spirit framework has been selected because it fulfills most of the principles of the implementation philosophy presented in Section 8.1.

Spirit is an object oriented recursive descent parser framework implemented using template meta-programming techniques. Expression templates allow to approximate the syntax of Extended Backus Normal Form (EBNF) EBNF (2003) completely in C++. Parser objects are composed through operator overloading and the result is a backtracking, top down parser that is capable of parsing rather ambiguous grammars.

The Spirit framework enables a target grammar to be written exclusively in C++. Inline EBNF grammar specifications can mix freely with other C++ code and, thanks to the generative power of C++ templates, are immediately executable.

The Spirit framework is part of the Boost C++ Libraries framework BOOST (2006), which consists on a set of free peer-reviewed portable and standard C++ source libraries. Boost libraries are intended to be widely useful, and usable across a broad spectrum of applications.

The functioning of a parser can be divided in three main steps: The *grammar definition*, the *matching procedure* and the *semantic actions*.

8.2.3.1 Grammar definition

The grammar definition consists of writing a set of rules for interpreting the input sequence. For instance, using the Spirit Parser framework, the corresponding C++ code for interpreting the following interval variable assignment

”`name=[real_number,real_number];`”

is presented in Table 8.1.

Table 8.1: Spirit grammar rule example.

```
rule<> assignment = name >> '=' >> interval >> ';' ;
rule<> name = alpha_p >> *(alnum_p | '_');
rule<> interval = '[' >> real_p >> ',' >> real_p >> ']';
```

`rule<>` is the Spirit type for defining a rule, `assignment`, `name` and `interval` are rule variables, `>>` is the sequencing operator to indicate that ”something” precedes ”something”, `alpha_p`, `alnum_p` and `real_p` are predefined Spirit rules (primitives) corresponding to an alphabetic character, alphanumeric character and a real numeric value. Finally, characters between ”” represents strings of characters and the *Kleene* star * indicates zero or more instances of ”something”. For example, the `name` rule can be read as: an alphabetic character followed or not by a sequence of alphanumeric characters with the `'_'` character (e.g. ”x123_4c”).

8.2.3.2 Matching procedure

The matching procedure consists on scanning the set of defined rules in order to see if one of them matches with a given command from the source code to be interpreted. For instance, given the following string,

”x=[1.5,7.8]”;

it matches with the `assignment` rule. Therefore, the matching procedure is said to be successful. On the other hand, the string

”x=(1.5,7.8)”;

does not match with the defined rules and the matching is failed.

8.2.3.3 Semantic actions

A semantic action is an action carried out when a matching is successful, being an action any implementable algorithm. In the Spirit Parser framework, semantic actions are placed after the corresponding rule and between brackets. For instance, a set of semantic rules for the already defined `assignment` rule is shown in Table 8.2.

Table 8.2: Semantic actions.

```
rule<> declaration = name >> '=' >> interval >> ';'[make_declaration] ;
rule<> name = alpha_p >> *(alnum_p | '_')[store_alnum];
rule<> interval = '[' >> real_p[store_real] >> ',' >>
real_p[store_real] >> ']';
```

Where `store_real`, `store_alnum` and `make_declaration` are semantic actions. For instance, the following sequence of actions could be performed:

- `[store_real]`: The two real values corresponding to the lower and upper bounds of the interval are stored in a stack.
- `[store_alnum]`: The alphanumeric string "x" is stored in a stack.
- `[make_declaration]`: The real values are recovered from the stack and an interval is created with them. The alphanumeric string is recovered from the stack. A new variable is created with the recovered character string and the interval. The stacks are cleared.

8.2.4 wxWidgets framework

With the purpose of creating a friendly windows based user interface, which at the same time respects the portability and open source criterions stated in Section 8.1, the wxWidgets framework wxWidgets (2006) has been chosen.

wxWidgets is an open source C++ Graphical User Interface (GUI) framework to make cross-platform programming. It allows to use one single source code on many different operative systems and platforms with very little (if any at all) code modifications, making the code and application portability as easy as it can get. Moreover, wxWidgets framework comes from with the Code::Blocks IDE.

8.3 FSTAR Solver

To implement the f^* algorithm presented in Chapter 3, numerical and symbolical programming techniques have been used. Concerning the numerical techniques, they have mainly been employed for implementing the modal interval arithmetic and the involved branch-and-bound algorithm. Concerning the symbolical techniques, they have been used for implementing the parser, and the algorithms for studying the monotony and optimality of the objective function.

The functioning of the FSTAR solver is summarized by the following steps:

1. The user writes the source code defining the problem in a text file using a predefined grammar.

2. A parser interprets the source code and executes the corresponding actions. Among other outputs, it generates a *binary tree* representing the objective function.

3. A symbolic differentiation algorithm differentiates the *binary tree* with respect to each incidence of the involved variables.

4. A symbolic algorithm studies the optimality of the objective function represented by the binary tree.

5. The *binary trees* are passed as parameters to the branch-and-bound algorithm and it is launched. The branch-and-bound algorithm uses the modal interval arithmetic library to evaluate the *binary trees*.

6. The obtained results are provided in a text file containing the computation time, the number of bisections carried out by the branch-and-bound algorithm and the achieved inner an outer approximations of the f^* extension with its corresponding tolerance.

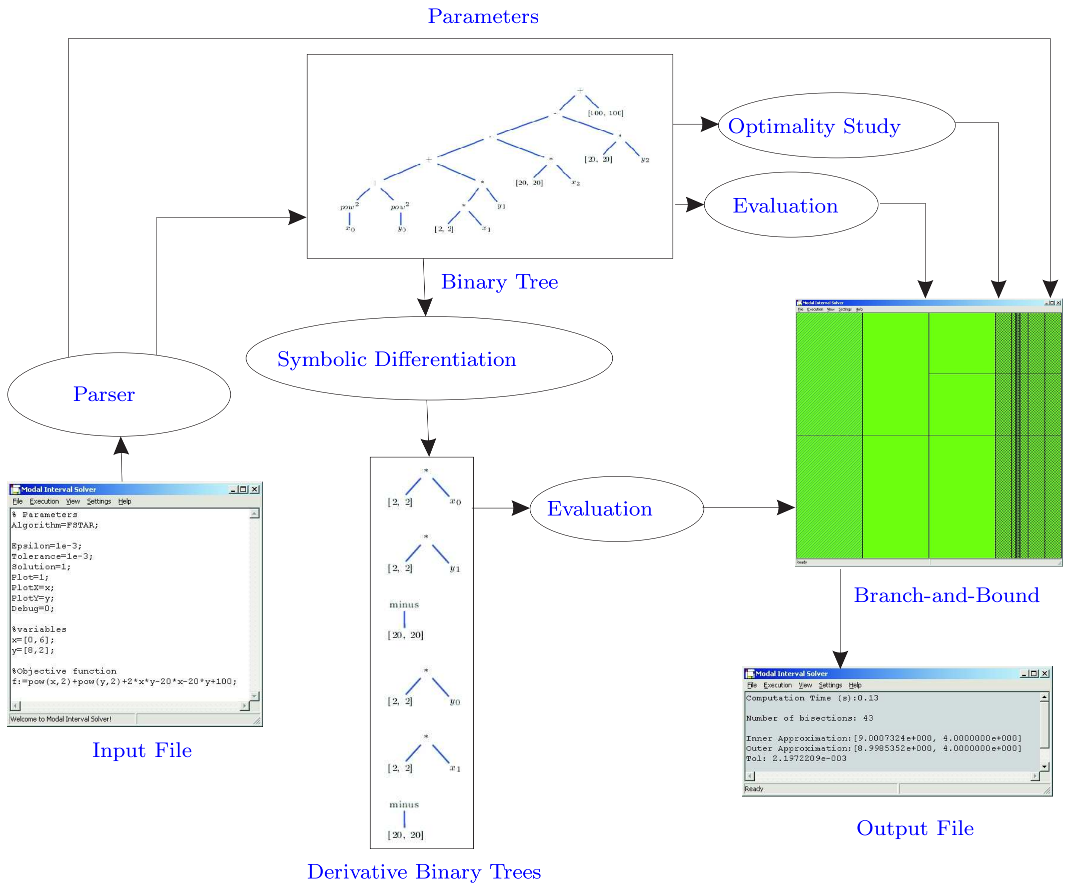
Figure 8.1 shows in a graphical way, the implementation architecture of the FSTAR solver.

8.3.1 Numeric implementation

As mentioned before, two different modules can be distinguished concerning the numeric implementation of the f^* algorithm: The modal interval arithmetic library and the branch-and-bound algorithm.

8.3.1.1 Modal interval arithmetic library

IvalDb (**I**nterval **Val**ue **Doub**le) is a C++ library which implements a set of modal interval arithmetic operators like $+, -, log, sin, etc$ taking advantage of the C++ operator overloading concept. The corresponding C++ class for its implementation has been named `ivalDb`. IvalDb library is inspired on an already existing single floating point precision modal interval library García Reyero & Martínez (1999), which control the numerical errors using floating point emulation . IvalDb doubles the floating point precision and uses a much more simple



```
% Parameters
Algorithm=FSTAR;

Epsilon=1e-3;
Tolerance=1e-3;
Solution=1;
Plot=1;
PlotX=x;
PlotY=y;
Debug=0;

%variables
x=[0,6];
y=[8,2];

%Objective function
f:=pow(x,2)+pow(y,2)+2*x*y-20*x-20*y+100;
```

Welcome to Modal Interval Solver!

```
Computation Time (s):0.13

Number of bisections: 43

Inner Approximation:[9.0007324e+000, 4.0000000e+000]
Outer Approximation:[8.9985352e+000, 4.0000000e+000]
Tol: 2.1972209e-003
```

Ready

Figure 8.1: FSTAR solver architecture.

strategy to control the numerical errors. IvalDb library is an original contribution of this thesis and is currently being extensively used.

IvalDb assures the numeric guaranty thanks to the use of FDLIBM Microsystems. (1996), a C library developed by Sun Microsystems. FDLIBM is a math C library which assures for an IEEE754 Kahan (1996) machine, in the worst case, an error of one bit for all the provided functions. Moreover, it assures multi-platform compatibility between IEEE754 machines. FDLIBM provides a function which allows to do upper and lower rounding of a floating point number. Then, knowing that the maximal error that can be committed by an FDLIBM function is one bit, it is easy to implement a guaranteed interval function by adding an ULP to the upper bound of the solution interval and by resting and ULP to the lower

Table 8.3: Modal interval arithmetic implementation example.

```
ivalDb exp(ivalDb &x)
{
double Inf=FDLIBM::exp(x.Infimum);  // FDLIBM evaluation
double Inf=FDLIBM::exp(x.Supremum);
ivalDb Result.SetBounds(Inf,Sup);
if(x.IsImproper()) return Dual(Round(Dual(Result)));
else return Round(Result);
}
```

Table 8.4: Round function implementation.

```
ivalDb Round(ivalDb &x)
{
FDLIBM::nextafter(x.Infimum,-1e-300);   //Rest ULP
FDLIBM::nextafter(x.Supremum,1e-300);   //Add ULP
}
```

bound of the solution interval. This method may be numerically conservative with respect other interval libraries but it is less time consuming because it does not change the rounding mode of the processor as other libraries usually do.

An example of implementation of a modal interval function using the FDLIBM library is shown is Table 8.3. Where, `FDLIBM::` is a *name-space* for the FDLIBM functions and `Round` is the function represented in Table 8.4.

An example of the ivalDb library utilization can be found in Table 8.5.

8.3.1.2 The branch-and-bound algorithm

For the implementation of the branch-and-bound algorithm involved in the f^* algorithm, six C++ classes have been created. Table 8.6 shows the list of C++

Table 8.5: IvalDb utilization example.

```
ivalDb x,y,z,r;          //Variables declaration
x=ivalDb(-5,5);          //variable assignment
y=ivalDb(10,-10);
z=ivalDb(6.5,15.3);
r=sin(x*exp(y))/z;       //Function evaluation
```

classes with a short description of their functionality and main attributes.

Table 8.6: Branch-and-bound algorithm C++ classes.

Class	Description
`twin`	It has two `ivalDb` attributes for representing the inner and outer approximations.
`incidence`	It has two `ivalDb` attributes representing the interval value and the monotony of the function with respect to the incidence.
`variable`	It contains a STL vector of `incidence` objects representing the incidences of the variable in the objective function. It also contains an `ivalDb` attributes representing the monotony of the function with respect to the variable.
`cell`	It contains two STL vectors of `variable` objects representing the involved proper variables and improper variables. It also contains a `twin` object representing the local inner and outer approximations of f^*.
`strip`	It contains a STL list of `cell` objects. It also contains a STL vector of `variable` objects representing the proper variables and a `twin` object representing the local inner and outer approximations of f^*.
`fstar`	Is the main class and contains a STL list of `strip` and a `twin` object representing the global inner and outer approximation of f^*. It also contains an attribute of the type `model` which contains the objective function and its derivatives.

8.3.2 Symbolic implementation

Modal Interval Analysis provides a set of theorems which are based on the monotony of the objective function and on the study of the optimality of its syntactic tree. Then, for a general implementation of the f^* algorithm, it is necessary to employ tools which allow to symbolically manipulate algebraic expressions. In computer science, one way to represent an algebraic expression is by means of a data structure called *binary tree*. For example, given the algebraic expression

$$f := x1 * u - x2 * pow(v,2) * sin(x1), \tag{8.1}$$

its binary tree is graphically represented in Figure 8.2, where the intermediate nodes of the tree represent the unary and binary operators and the terminal nodes (the leaves of the tree) represent the variables.

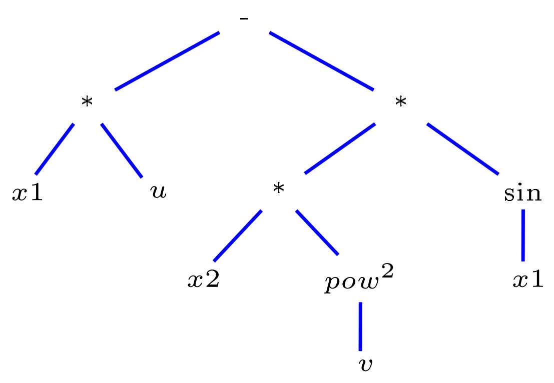


Figure 8.2: Binary tree representation.

Three main routines conforms the symbolic implementation:

- A parser, based on the Spirit framework, which provides a specific grammar to introduce the problem definition. As output, it generates a binary tree representing the objective function.
- A symbolic differentiation routine, which takes as input the binary tree and computes its derivatives with respect to each incidence of the variables.
- A routine to study the optimality of the objective function, which is also based on the study of the binary tree.

8.3.2.1 The parser

The specific grammar for introducing a problem in the FSTAR solver is conformed by the set of commands from Table 8.7

For example, a problem definition using the grammar from Table 8.7 is shown in Table 8.8 and its corresponding output file in Table 8.9.

As mentioned before, the output of the parser is a binary tree representing the objective function. For the implementation of the binary tree, the three C++ classes described in Table 8.10 have been created.

For the example of Table 8.8, a graphical representation of the resulting binary tree, where the subindexes represent the different incidences of the variables, can be observed in Figure 8.3.

Table 8.7: FSTAR solver grammar.

Command	**Description**
`Algorithm=FSTAR;`	Select the Logic Solver.
`%Commentaries`	Commentaries started with %.
`Tolerance=real_number;`	Desired precision for the approximation of f^*.
`Epsilon=real_number;`	Interval size from which the bisection procedure does not bisect a variable.
`var_name = [lb,hb];`	Variable declaration and interval assignment.
`f:=f(var_name);`	Objective function definition where, f is a non-linear function involving the declared variables and the function symbols: +, -, *, /, exp, log, cos, sin, tan, acos, asin, atan, pow(x,n), sqrt, min, max, abs.

Table 8.8: FSTAR solver grammar example.

```
Algorithm=FSTAR;
Tolerance=1e-3;
Epsilon=1e-3;
x = [0,6];
y = [8,2];
f:=pow(x,2)+pow(y,2)+2*x*y-20*x-20*y+100;
```

Table 8.9: FSTAR solver output.

```
Computation Time (s):0.15
Number of bisections: 43
Inner Approximation:[9.0007324e+000, 4.0000000e+000]
Outer Approximation:[8.9985352e+000, 4.0000000e+000]
Tol: 2.1972209e-003
```

The algebraic expression represented by the resulting binary tree can be easily evaluated by means of a simple recursive procedure which crosses over the tree and stores the partial results in the intermediate nodes. Figure 8.4 shows the same binary tree after its evaluation to obtain an outer approximation of $f^*(X, Y)$ where $X = [0, 6]$ and $Y = [8, 2]$. Notice the t transformation applied over the Y variable.

Table 8.10: Binary tree C++ classes.

Class	**Description**
`node`	Represents a `node` in the binary tree. It contains two pointers to two different `node` objects and an `ivalDb` object used for the tree evaluation.
`tree`	Represents the binary tree. Contains a `node` pointer to the root of the binary tree and a list of `node` pointers to the root of the binary trees representing the derivatives. It also contains a vector of `node` objects corresponding to the terminal nodes of the tree.
`model`	It contains a `tree` object, a **cell** object and a vector of `variable` pointers to the `cell` object to be evaluated.

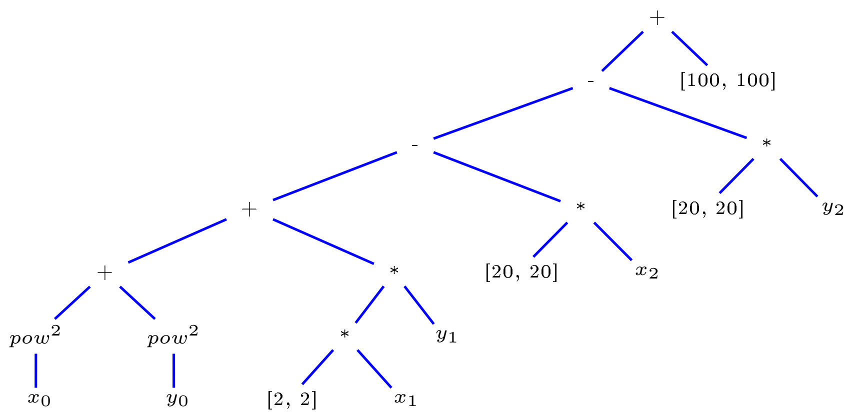


Figure 8.3: Binary tree representation for example from Table 8.8.

8.3.2.2 Symbolic differentiation

As mentioned in Chapter 3, the f^* algorithm requires from the monotony of the objective function to apply the corresponding modal interval theorems. Then, a differentiation algorithm, which is based on the chain rule, has been developed for this purpose. This algorithms is a recursive procedure which crosses over the binary tree and applies the corresponding differentiation rule to each node. As output, it produces a set of binary trees representing each one of the derivatives of the objective function with respect to the incidences of each variable. A simplification routine is also provided in order to simplify the resulting trees from the differentiation routine. This simplification is important because it can notably reduce the size of the binary trees and consequently reduces the computation time. For instance, for the problem of Table 8.8, the resulting binary trees representing

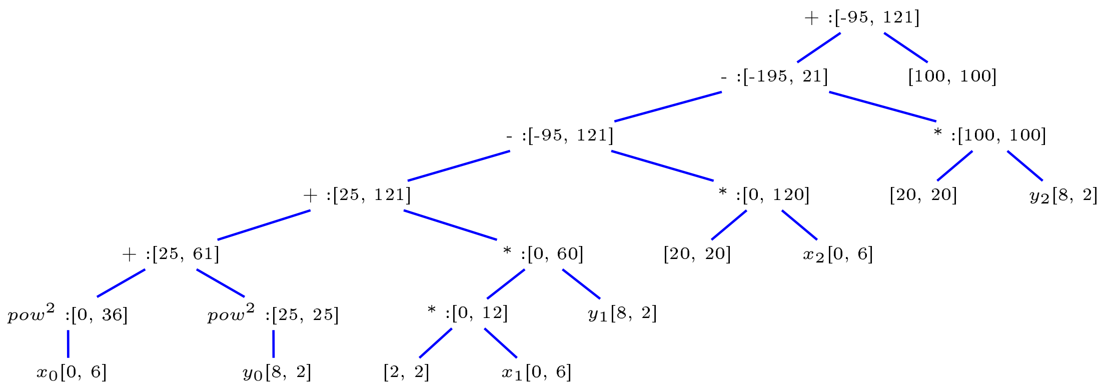


Figure 8.4: Binary tree after its evaluation.

the corresponding derivatives are observed in Figure 8.5.

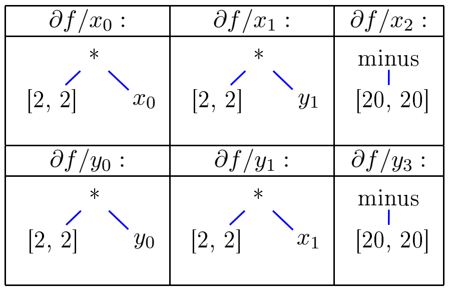


Figure 8.5: Derivatives binary trees.

Then, evaluating the derivative trees, the monotony of the function with respect to each incidence and the variables can be obtained.

8.3.2.3 Tree-optimality study

In order to study the tree-optimality of the objective function, a recursive algorithm has been implemented. This algorithm crosses over the binary tree and looks for the non-uniformly monotonic operators (e.g. $*,/$). If the branches which are followed downwards in the syntactic tree only involve one-variable operators, the syntactic tree is optimal, if not, these branches are evaluated and if they are both positive or negative, the syntactic tree is also optimal. If the branches are none monotonic, nothing can be assured and the tree is considered non-optimal.

For example, given the continuous function $f(x, y, z, u) = (x + y) * (z + u)$, it is not tree optimal because it has a non-uniformly monotonic operator $(*)$ which is followed downwards by binary operators $(+)$. However, for the domains $x \in [2, 6]'$, $y \in [2, 10]'$, $z \in [1, 5]'$ and $u \in [2, 4]'$, the evaluation of the downwards branches is positive, as can be observed in Figure 8.6. Then, the function is tree-optimal.

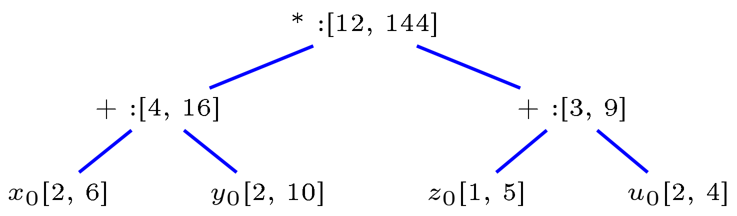


Figure 8.6: Tree-optimality study representation.

8.4 QRCS Solver

The Quantified Real Constraint Satisfaction (QRCS) solver is the software implementation of the modal interval inclusion test defined in Section 3.2.1.8, for proving the satisfiability of a class of quantified real constraints. QRCS solver has the same implementation architecture than the FSTAR solver presented in Section 8.3. It also takes as input a text file with the problem definition and provides as output another text file with the computation time, the number of carried out bisections by branch-and-bound algorithm, the consistency of the logical formula and the inner and outer approximations of the corresponding f^* computation.

The specific grammar for introducing the problem definitions to the QRCS solver is defined by the set of commands presented in Table 8.11.

For example, a problem definition using the grammar from Table 8.11 is shown in Table 8.12 and its corresponding output file in Table 8.13.

Table 8.11: QRCS solver grammar.

Command	**Description**
`Algorithm=QRCS;`	Select the Logic Solver.
`%Commentaries`	Commentaries started with `%`.
`Tolerance=real_number;`	Desired precision for the approximation of f^*.
`Epsilon=real_number;`	Interval size from which the bisection procedure does not bisect a variable.
`U(var_name,[lb,hb]);`	Universally quantified variable with its associated interval.
`E(var_name,[lb,hb]);`	Existentially quantified variable with its associated interval.
`lf:=f(var_name)op 0;`	Logical formula definition where, f is any non-linear function involving the declared variables and the function symbols: +, -, *, /,exp, log, cos, sin, tan, acos, asin, atan, pow(x,n), sqrt, min, max, abs, and op is the relational operators =, >, <, $\geq$ or $\leq$.

Table 8.12: QRCS solver grammar example.

```
Algorithm=QRCS;
Epsilon=0.01;
Tolerance=0.01;
% Variables
U(x,[0,6]);
U(z,[5,8]);
E(y,[2,8]);
% Constraint
lf:=pow(x,2)+pow(y,2)+2*x*y-20*x-20*y+100-z=0;
```

Table 8.13: QRCS solver output.

```
Computation Time (s):0.03
Number of bisections: 13
Consistency: True
Inner Approximation:[2.9726562e+000, -1.0000000e+000]
Outer Approximation:[2.6562500e-001, -1.0000000e+000]
Tol: 2.7070313e+000
```

8.5 MINIMAX Solver

The MINIMAX solver is the software implementation of the continuous minimax optimization algorithm presented in Chapter 3. It uses almost the same implementation architecture as the FSTAR solver presented in Section 8.3. Only some small modifications have been introduced to adapt it.

Concerning the solver input, it also consists on a text file containing a set of commands defining the minimax optimization problem to be solved. As output, the MINIMAX solver produces a text file with the computation time, the number of bisections carried out by branch-and-bound algorithm, an interval containing the minimax value with its corresponding tolerance and a list of cells which are candidates to contain the minimax points.

The specific grammar for introducing a problem into the MINIMAX solver is conformed by the set of commands given in Table 8.14.

Table 8.14: MINIMAX solver grammar.

Command	**Description**
`Algorithm=MINIMAX;`	Select the MINIMAX Solver.
`%Parameters`	Commentaries started with `%`.
`Tolerance=real_number;`	Desired precision for the approximation of f^*.
`Epsilon=real_number;`	Interval size from which the bisection procedure does not bisect a variable.
`MIN(var_name,[lb,hb]);`	Variable to minimize with its associated interval.
`MAX(var_name,[lb,hb]);`	Variable to minimize with its associated interval.
`f:=f(var_name);`	Objective function definition where, `f` is any non-linear function involving the declared variables and the function symbols: +, -, *, /,exp, log, cos, sin, tan, acos, asin, atan, pow(x,n), sqrt, min, max, abs.
`c:=f(var_name)op 0;`	Constraint definition where, `op` is the relational operators $>$, $<$, $\geq$ or $\leq$.

For example, a minimax optimization problem definition using the grammar from Table 8.14 is shown in Table 8.15 and its corresponding output file in Table 8.16.

Table 8.15: MINIMAX grammar example.

```
Algorithm=MINIMAX;
Epsilon=1e-3;
Tolerance=1e-3;
MIN(x,[-3.1416,3.1416]);
MAX(y,[-3.1416,3.1416]);
f:=pow(cos(y)+cos(2*y+x),2);
c:=y-x*(x+2*3.1416)<0;
c:=y-x*(x-2*3.1416)<0;
```

Table 8.16: MINIMAX solver output.

```
Computation Time (s):1.733
Number of bisections: 289
MinMax Approximation:[-1.5186201e-001, -1.4563311e-001]
Tol: 6.2288993e-003
Minimax List: (Number of Boxes: 16)
box----------------------------------------------------
x=[-5.5223438e-001, -5.3996250e-001]
y=[-3.1416000e+000, -3.1170563e+000]
Minimax:[-1.4865319e-001,,-1.3288647e-001]
---------------------------------------------------box
...
```

8.6 QSI Solver

The QSI solver is the software implementation of the Quantified Set Inversion (QSI) algorithm presented in Chapter 4. Two different modules can be mainly differentiated in its implementation: a branch-and-bound algorithm and the modal interval inclusion test, which corresponds to the QRCS solver presented in Section 8.4.

The sequence of actions carried out by the QSI solver is summarized in the following steps:

1. The user introduces the problem definition in a text file using the grammar from Table 8.18.

2. A parser interprets the text file and generates a *binary tree* for each one of the involved constrains.

3. A symbolic differentiation algorithm differentiates the binary trees with respect to each incidence of the involved variables.

4. A symbolic algorithm studies the optimality of the involved constrains represented by the binary trees.

5. The branch-and-bound algorithm is launched. For each one of the resulting boxes, the QRCS solver is used to test their consistency.

6. The obtained result is provided in a text file containing the computation time, the number of bisections, a box containing the solution set and denoted by `Hull`, and three lists of boxes, one of solution boxes, another of non-solution boxes and a last one of undefined boxes. QSI solver also provides a graphical output consisting on a two dimensional projections of the generated solution.

For its implementation, the three extra C++ classes from Table 8.17 have been created.

Table 8.17: QSI solver C++ classes.

Class	**Description**
`box`	Vector of `ivalDb` objects which represents a vector of free-variables
`csp`	Vector of `fstar` objects representing the involved constraints.
`qsi`	Main class containing a `csp` object and a STL list of `box` objects.

Table 8.18 shows the corresponding grammar for introducing a problem into the QSI solver.

Table 8.19 shows an example of a QSI problem definition using the grammar from Table 8.18 and Table 8.20 shows its corresponding output file.

For the problem of Table 8.19, the corresponding graphical output can be observed in Figure 8.7. Where red boxes are boxes contained in the solution set, blue boxes are outside of the solution set and green boxes are undefined boxes.

Table 8.18: QSI solver grammar.

Command	**Description**
`Algorithm=QSI;`	Select the QSI Solver.
`%Parameters`	Commentaries started with `%`.
`QSIEps=real_number;`	Interval size from which the bisection procedure of the QSI algorithm does not bisect a variable.
`Tolerance=real_number;`	Desired precision for the approximation of f^*.
`Epsilon=real_number;`	Interval size from which the bisection procedure of the f^* algorithm does not bisect a variable.
`PlotX=x;`	Allows to select the free variable to be plotted on the x axis (graphical output).
`PlotY=y;`	Allows to select the free variable to be plotted on the y axis (graphical output).
`F(var_name,[lb,hb]);`	Free variable with its associated interval.
`U(var_name,[lb,hb]);`	Universally quantified variable with its associated interval.
`E(var_name,[lb,hb]);`	Existentially quantified variable with its associated interval.
`c:=f(var_name)op 0;`	Constraint definition where, `f(var_name)` is a continuous function and `op` is the operators $>$, $<$, $\geq$ or $\leq$.

Table 8.19: QSI solver grammar example.

```
Algorithm=QSI;
Epsilon=1e-2;
Tolerance=1e-2;
CSPEps=0.01;
PlotX=x;
PlotY=y;
% Variables
F(x,[-2,2]);
F(y,[-2,2]);
E(z,[-2,2]);
% Constraints
c:=pow(x,2)+pow(y,2)+pow(z,2)-1=0;
```

8.7 SQUALTRACK Solver

The SQUALTRACK solver is the software implementation of the fault detection algorithm presented in Chapter 6 and developed in the context of the European

Table 8.20: QSI solver output.

```
Solution File:
Computation Time(s):1.763
Bisections:707
Hull:
x=[-0.9688, 0.9688]
y=[-0.9688, 0.9688]
Solution Boxes:
Box:
x=[-0.5, 0]
y=[-0.5, 0]
...
Non Solution Boxes:
Box:
x=[-2, -1]
y=[-2, -1]
...
Undefined Boxes:
Box:
x=[-0.03125, 0]
y=[-1.031, -1]
...
```

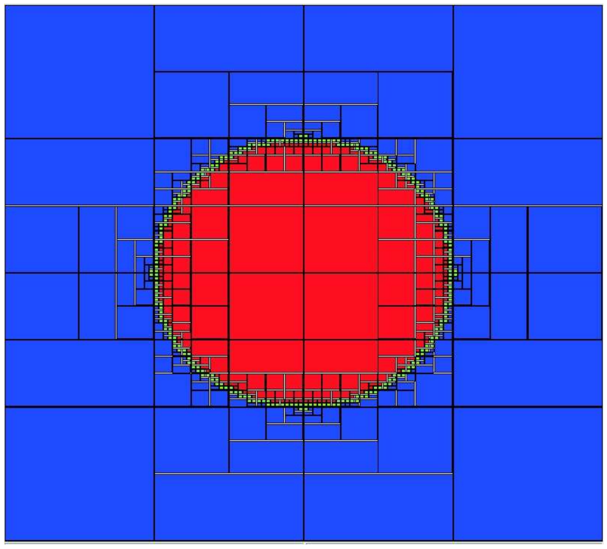


Figure 8.7: QSI graphical output.

CHEM project CHEM Consortium (2000). Figure 8.7 shows the icon of the developed software for the CHEM project.

Figure 8.8: SQUALTRACK solver icon.

The SQUALTRACK solver can be decomposed into the following modules:

- A parser based on the Spirit framework (see Section 8.2.3.
- A symbolic algorithm, which builds the Analytical redundancy Relations (ARRs) for each window length.
- A Communication Process Interface (CPI) which can read data from a text file or from a real process.
- The QRCS solver presented in Section 8.4.
- The fault detection algorithm sketched in Section 6.2.3.

For its implementation, the three C++ classes from Table 8.17 have been created.

The sequence of actions carried out by the SQUALTRACK solver is summarized by the following steps:

1. The user introduces the problem definition in a text file using the grammar from Table 8.22.

Table 8.21: SQUALTRACK solver C++ classes.

Class	Description
`data`	Vector of `string`, `double` and `ivalDb` objects which represents the process data.
`squaltrack`	Main class containing the fault detection routine.
`sqtchart`	Class for representing the graphical output based on the wxWidgets framework.

2. A parser interprets the input text file and generates, among other structures, a binary tree representing the analytical redundancy relation (ARR) for a time window length of 1.

3. A symbolic routine builds the analytical redundancy relations (ARRs) for each one of the selected time window lengths.

4. A symbolic differentiation routine differentiates the ARRs with respect to each incidence of the involved variables.

5. A symbolic routine studies the optimality of the resulting ARRs.

6. The fault detection routine is launched. For each sample time, the process data is read and assigned to the corresponding ARRs. The ARRs are evaluated using the QRCS solver.

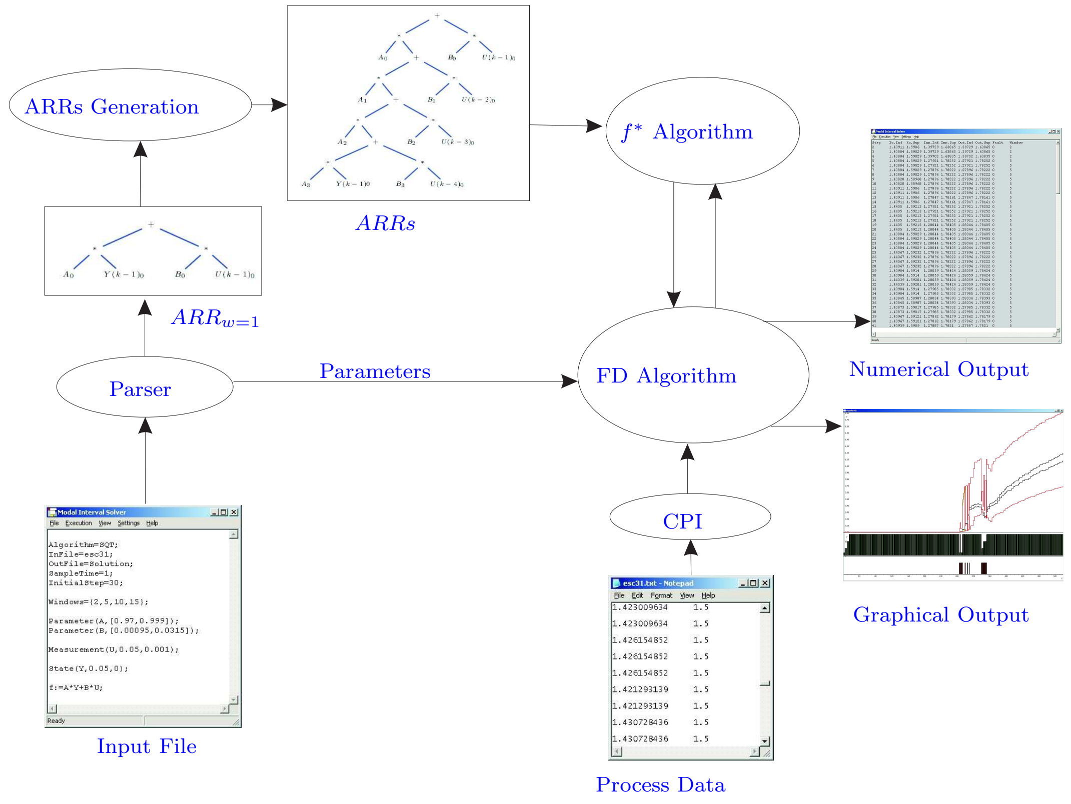
7. The obtained result is provided in a numerical and graphical way through the user interface.

Figure 8.9 graphically shows the SQUALTRACK solver architecture.

8.7.1 The parser

A parser, based on the Spirit framework, provides the set of commands given in Table 8.22.

An example of a SQUALTRACK problem definition using the grammar from Table 8.22 is shown in Table 8.23.



```
Algorithm=SQT;
InFile=esc31;
OutFile=Solution;
SampleTime=1;
InitialStep=30;

Windows={2,5,10,15};

Parameter(A,[0.97,0.999]);
Parameter(B,[0.00095,0.0315]);

Measurement(U,0.05,0.001);

State(Y,0.05,0);

f:=A*Y+B*U;
```

```
1.423009634     1.5
1.423009634     1.5
1.426154852     1.5
1.426154852     1.5
1.426154852     1.5
1.421293139     1.5
1.421293139     1.5
1.430728436     1.5
1.430728436     1.5
```

Figure 8.9: SQUALTRACK solver architecture.

8.7.2 Analytical Redundancy Relations construction

The output of the parser produces a binary tree corresponding to the function model of length 1, then, it is necessary to automatically generate the binary tree for the analytical redundancy relations (ARRs) corresponding to each one of the introduced window lengths. This task is carried out through the manipulation of the binary tree obtained with the parser. The implemented algorithm crosses over the binary tree and looks for the different occurrences of the output variable. Then, it recursively substitutes these occurrences by the same binary tree but with the corresponding indexes for the new introduced variables and occurrences. For example, given the following generic first order model

$$Y(k) = A * Y(k-1) + B * U(k-1), \tag{8.2}$$

where $Y(k)$ and $Y(k-1)$ are the output variable at time instants k and $k-1$, $U(k-1)$ is the input variable and, A and B are model parameters. The graphical

Table 8.22: SQUALTRACK solver grammar.

`Algorithm=SQT;`	Select the SQUALTRACK Solver.
`InFile=input_file_name;`	Is the input file name from where the process data is read. The data must be organized by columns and the first row must contain the name of the variables.
`OutFile=output_file_name;`	Is the output file name where the numerical results are written.
`SampleTime=ts;`	Is the maximum available computing time between two samples.
`Windows={w1,w2,...,wn};`	Is the set of window lengths.
`Parameter(name,interval);`	Is a model parameter declaration where `name` is the parameters name and `interval` is the corresponding interval.
`Measurement(name,r_noise,a_noise);`	Is a measured model input declaration where, `name` is the name of the measurements which must correspond with the one of the input file, `r_noise` is the relative noise expressed in percentage and `a_noise` is the absolute noise.
`State(name,r_noise,a_noise);`	Is the measured model output declaration where `name` is the name of the measurements which must correspond with the one of the input file.
`f:=F(Parameter,Measurements,State);`	Is any non-linear function involving the declared parameters, measurements and the state.

representation of the binary tree corresponding to the right part of the model equation is represented in Figure 8.10.

where the subindexes represents the occurrences of the variables. Then, the corresponding binary tree for a window length of 5 is represented in Figure 8.11.

8.7.3 Communication Process Interface

SQUALTRACK solver is presented in two different versions, one for working with offline data and another for working with online data. Then, the communication process interface (CPI) is different for each one of these versions.

Table 8.23: SQUALTRACK grammar example.

```
Algorithm=SQT;
%Parameters
InFile=input_file_name;
OutFile=output_file_name;
SampleTime=1;
Windows={1,5,25,50};

%Model parameters
Parameter(A,[1,2]);
Parameter(B,[3,4]);

%Measurements
Measurement(U,0.05,0);

%State variable
State(Y,0.05,0);

%Function model
f:=A*Y+B*U;
```

8.7.3.1 Offline CPI

The off-line version of the SQUALTRACK solver obtains the required process data from a text file. For this purpose, standard C++ tools for reading/writing on text files and manipulating character strings have been used. The unique restriction about the data file provided by the user the ordering of the information. The data has to be ordered by columns and separated by space characters. In the first row of the file, the names of the variables, which must correspond with

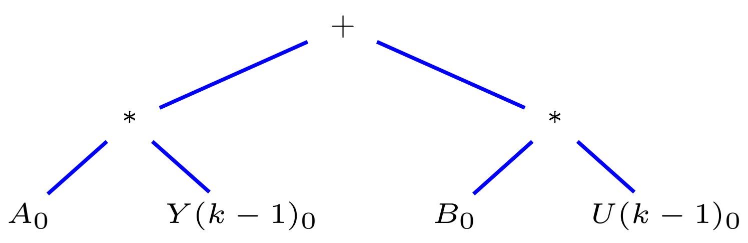


Figure 8.10: First order model representation.

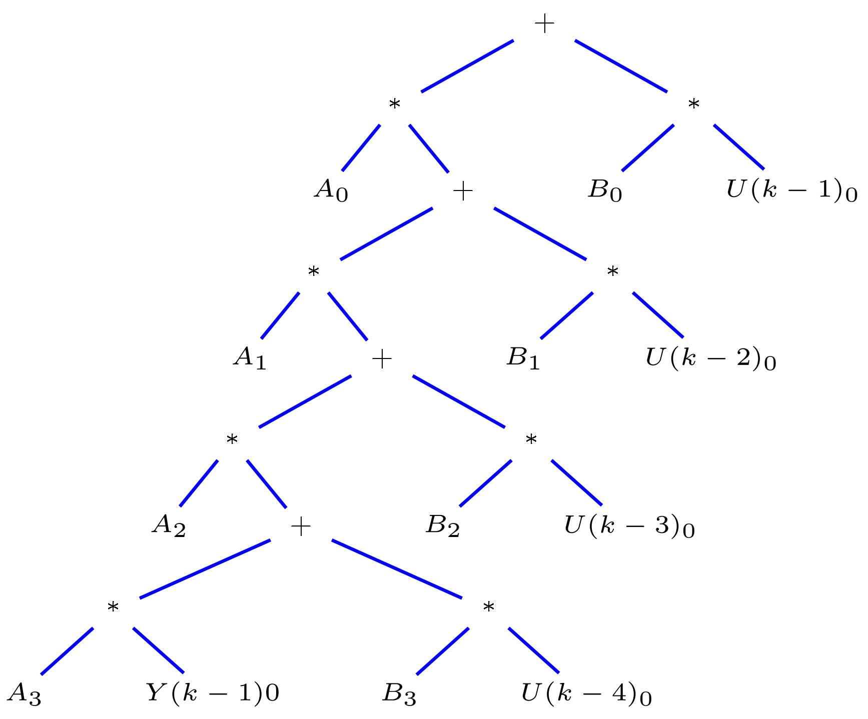


Figure 8.11: First order model representation for a window length of 5.

the names of the variables declared in the problem definition, must be provided. Finally, the file must have a ".txt" extension. Table 8.24 shows an example of the corresponding data file for the problem example of Table 8.23.

Table 8.24: SQUALTRACK data input file.

Y	U
1.514855214	1.5
1.514563286	1.5
1.514563286	1.5
1.514563286	1.5
1.513979551	1.5
1.514855214	1.5
1.514855214	1.5
...	

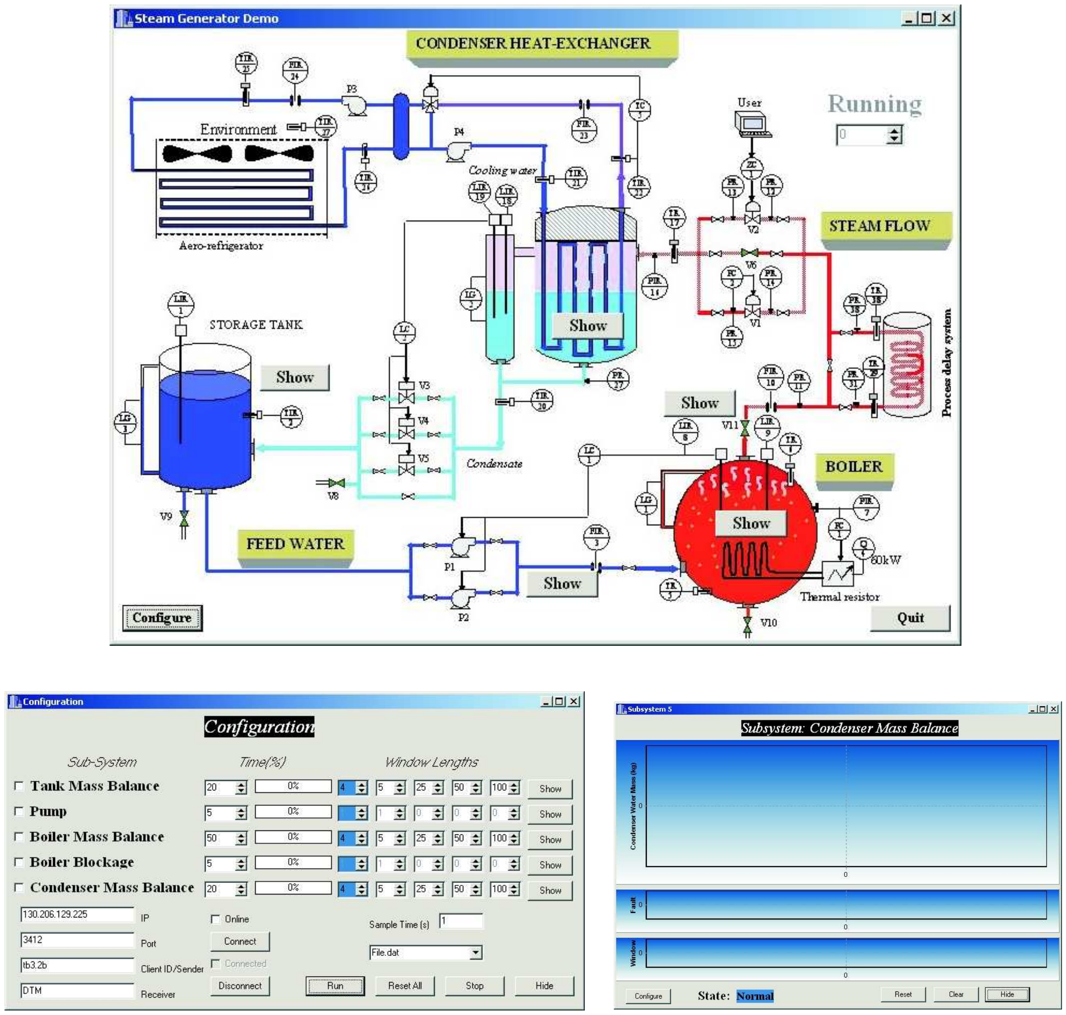
8.7.3.2 Online CPI

In the context of the CHEM project CHEM Consortium (2000), an online version of the SQUALTRACK solver has been implemented. This version of the solver is provided with a communication process interface, developed by the CHEM project, which allows to communicate with the process and with other software tools devoted to solve other supervision tasks (e.g. diagnosis, re-scheduling). This version of the solver has been implemented using Borland C++ Builder Builder (2006) for compatibility reasons with the provided CPI. Figure 8.12 shows some snapshots of the online version of the SQUALTRACK solver developed for The Steam Pilot Plant at the *Laboratoire d'Automatique et d'Informatique Industrielle de Lille* (France). This version of the SQUALTRACK solver allows to simultaneously supervise various subsystems of the process. By introducing a percentage, it is possible to assign the computation time dedicated to each subsystem. A main windows representing the whole process allows to easily visualize if any of the subsystems is behaving faulty or not. It is also possible to visualize the envelopes generated for each subsystem. A configuration window is also provided, which allows to introduce the required parameters for the communication interface and other configuration parameters concerning the execution.

8.7.4 Numerical and graphical outputs

The SQUALTRACK solver generates a numerical and a graphical output. The numerical output consists on a text file containing the time step (Step), the output variable (Xc), the inner (Inn) and outer (Out) approximations of the f^* computation, a boolean variable (Fault) representing the fault and the biggest used window length (Window) for each time step. Table 8.25 shows an example of this numerical output.

In order to provide a more friendly output to the final user, the SQUALTRACK solver plots to the graphical user interface (GUI) the computed inner and an outer approximations of the model output together with the measured output of the system. As all these signals are represented by intervals, it is easy to see if the outer approximation intersects or not with the measured output. Then,



Configuration

Sub-System	Time(%)		Window Lengths					
☐ Tank Mass Balance	20	0%	4	5	25	50	100	Show
☐ Pump	5	0%		1	0	0	0	Show
☐ Boiler Mass Balance	50	0%	4	5	25	50	100	Show
☐ Boiler Blockage	5	0%		1	0	0	0	Show
☐ Condenser Mass Balance	20	0%	4	5	25	50	100	Show

IP: 130.206.129.225
Port: 3412
Client ID/Sender: tb3.2b
Receiver: DTM
☐ Online
☐ Connected
Sample Time (s): 1
File.dat

Connect | Disconnect | Run | Reset All | Stop | Hide

Figure 8.12: SQUALTRACK solver online version.

Table 8.25: SQUALTRACK numerical output.

```
Step    Xc.Inf  Xc.Sup  Inn.Inf Inn.Sup Out.Inf Out.Sup Fault   Window
2       1.43911 1.5906  1.39729 1.63865 1.39729 1.63865     0   2
3       1.43884 1.59029 1.39729 1.63865 1.39729 1.63865     0   2
4       1.43884 1.59029 1.39702 1.63835 1.39702 1.63835     0   2
5       1.43884 1.59029 1.27921 1.78252 1.27921 1.78252     0   5
6       1.43884 1.59029 1.27921 1.78252 1.27921 1.78252     0   5
7       1.43884 1.59029 1.27896 1.78222 1.27896 1.78222     0   5
...
```

it is possible to determine in a visual way if the process is behaving normally or faulty.

Figure 8.13 shows the graphical output of the SQUALTRACK solver. The upper graph shows the approximations (inner in green and outer in red) for the output variable and the corresponding measurement (in black). Note that often inner and outer approximations are not graphically distinguishable because they are very close. The graph in the middle indicates the longest window length that has been used at each time step by means of green bars. Finally, the lower graph shows a red bar when a fault is detected.

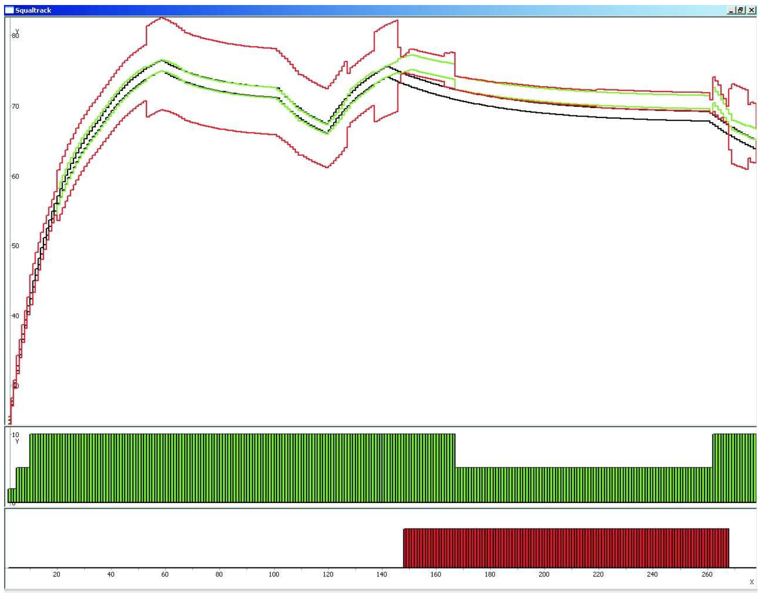


Figure 8.13: SQUALTRACK solver graphical output.

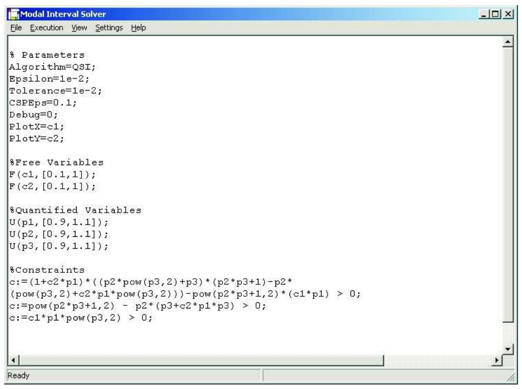
8.8 Generic user interface

All the solvers presented in this chapter have been centralized into a unique user interface, completely developed using the wxWidgets framework and which has been referred as Modal Interval Solver (MISO). This GUI allows to introduce the problems and to obtain the corresponding numerical and graphical outputs. A problem definition can be directly introduced by taping the commands in the main window using the corresponding grammar or by loading it from a text file using the window menu option `File->Open`. It is also possible to save the realized changes by selecting `File->Save` or to save the problem to a text file with a different name selecting `File->Save as`. To quit the application, the `File->Quit` option has to be selected. Once the problem definition has been introduced, it can be executed by selecting `Execution->Run`. During the execution it can be paused (`Execution->Pause`) or interrupted (`Execution->Break`). Once the execution is terminated, the output file automatically appears in the main window. It is possible to switch between the input file and the output file using the options (`View->Output/View->Input`). Solvers which dispose of a graphical output automatically show it during the execution. Figure 8.14 shows different snapshots of the MISO user interface.

8.9 Modal Interval Remote Solver

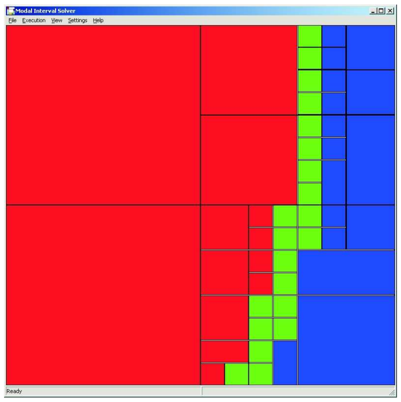
With the purpose of spreading the use of MIA based solvers, a web page allowing to remotely use them from a PC with an Internet connection, has been created. The main advantage of using a centralized system is that it guaranties that all the users are using the last updated version. Another interesting property is the information feedback obtained from the users, which allows to continuously improve the solvers. The web application has been developed by Flórez from MiceLab laboratory, using the Common Gateway Interface (CGI) standard (CGI), HTML language HTML (2006) and the PERL programming language Perl (2006).

The functioning of the Modal Interval Remote Solver(MIRS) is summarized in the following steps.

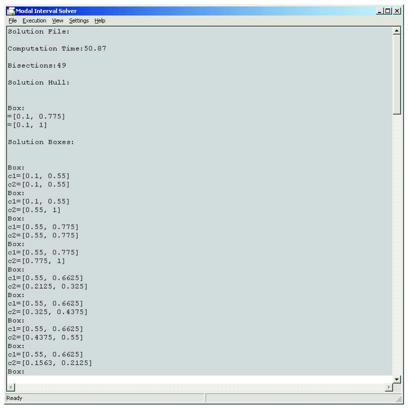


```
% Parameters
Algorithm=QSI;
Epsilon=1e-2;
Tolerance=1e-2;
CSPEps=0.1;
Debug=0;
PlotX=c1;
PlotY=c2;

%Free Variables
F(c1,[0.1,1]);
F(c2,[0.1,1]);

%Quantified Variables
U(p1,[0.9,1.1]);
U(p2,[0.9,1.1]);
U(p3,[0.9,1.1]);

%Constraints
c:=(1+c2*p1)*((p2*pow(p3,2)+p3)*(p2*p3+1)-p2*
(pow(p3,2)+c2*p1*pow(p3,2)))-pow(p2*p3+1,2)*(c1*p1) > 0;
c:=pow(p2*p3+1,2) - p2*(p3+c2*p1*p3) > 0;
c:=c1*p1*pow(p3,2) > 0;
```

Input File

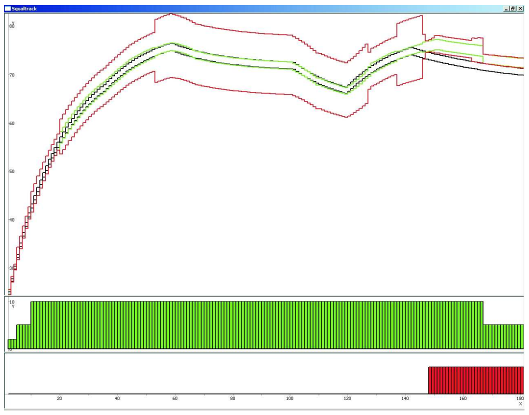


QSI Graphical Output



```
Solution File:

Computation Time:50.87

Bisections:49

Solution Hull:


Box:
=[0.1, 0.775]
=[0.1, 1]

Solution Boxes:


Box:
c1=[0.1, 0.55]
c2=[0.1, 0.55]
Box:
c1=[0.1, 0.55]
c2=[0.55, 1]
Box:
c1=[0.55, 0.775]
c2=[0.55, 0.775]
Box:
c1=[0.55, 0.775]
c2=[0.775, 1]
Box:
c1=[0.55, 0.6625]
c2=[0.2125, 0.325]
Box:
c1=[0.55, 0.6625]
c2=[0.325, 0.4375]
Box:
c1=[0.55, 0.6625]
c2=[0.4375, 0.55]
Box:
c1=[0.55, 0.6625]
c2=[0.1563, 0.2125]
Box:
```

Numerical Output File

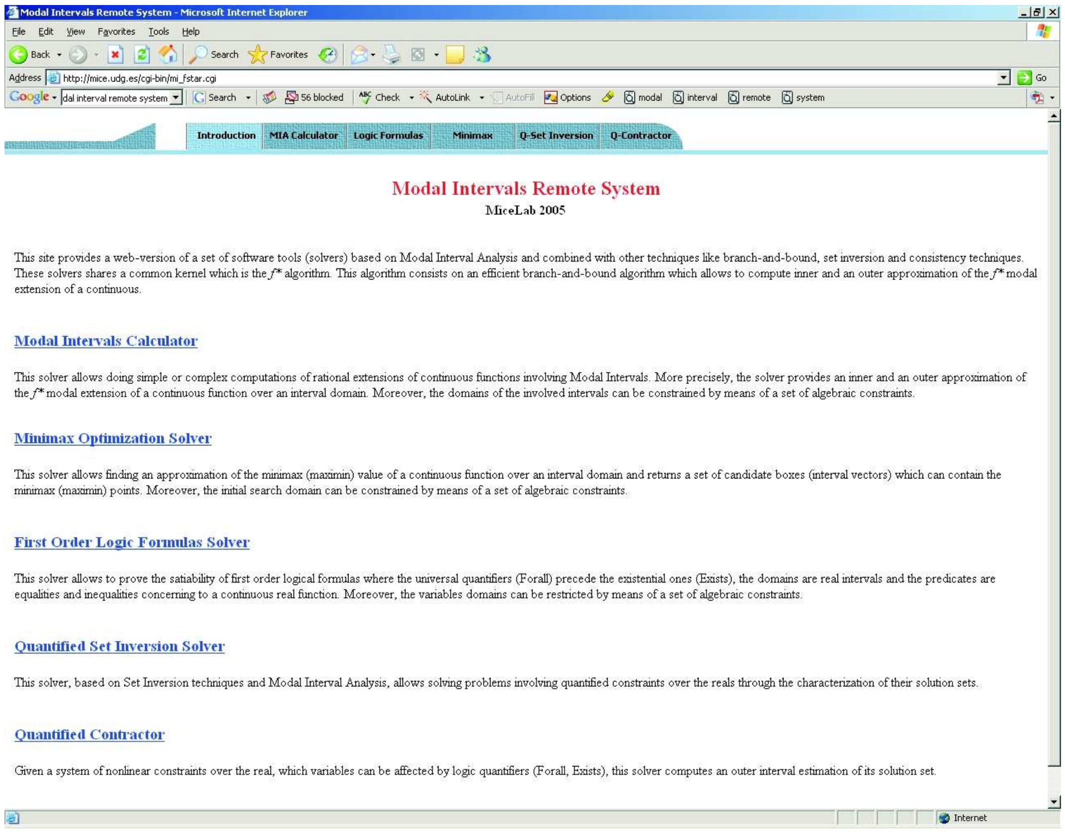


SQUALTRACK Graphical Output

Figure 8.14: Modal Interval Solver graphical user interface.

1. The user access from a web browser to the following URL MiceLab (2005) and chooses from the main page which solver he wants to use. This main page provides a short description of the available solvers. Figure 8.15 shows the main page of the MIRS.

2. Once the solver has been selected, the user can introduce the problem in the available text boxes. Instructions about how to introduce the problems are provided in the same web page by clicking the Help button. It is also possible to load a previously defined problem from a stored text file by clicking the Browse button. The user can also select different examples from a *combo box*. Figure 8.16 a snapshot of the MIRS corresponding to the QSI solver.


Modal Intervals Remote System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://mice.udg.es/cgi-bin/mi_fstar.cgi Go
Google dal interval remote system Search 56 blocked Check AutoLink AutoFill Options modal interval remote system
Introduction MIA Calculator Logic Formulas Minimax Q-Set Inversion Q-Contractor
Internet

Modal Intervals Remote System

MiceLab 2005

This site provides a web-version of a set of software tools (solvers) based on Modal Interval Analysis and combined with other techniques like branch-and-bound, set inversion and consistency techniques. These solvers shares a common kernel which is the *f** algorithm. This algorithm consists on an efficient branch-and-bound algorithm which allows to compute inner and an outer approximation of the *f** modal extension of a continuous.

Modal Intervals Calculator

This solver allows doing simple or complex computations of rational extensions of continuous functions involving Modal Intervals. More precisely, the solver provides an inner and an outer approximation of the *f** modal extension of a continuous function over an interval domain. Moreover, the domains of the involved intervals can be constrained by means of a set of algebraic constraints.

Minimax Optimization Solver

This solver allows finding an approximation of the minimax (maximin) value of a continuous function over an interval domain and returns a set of candidate boxes (interval vectors) which can contain the minimax (maximin) points. Moreover, the initial search domain can be constrained by means of a set of algebraic constraints.

First Order Logic Formulas Solver

This solver allows to prove the satiability of first order logical formulas where the universal quantifiers (Forall) precede the existential ones (Exists), the domains are real intervals and the predicates are equalities and inequalities concerning to a continuous real function. Moreover, the variables domains can be restricted by means of a set of algebraic constraints.

Quantified Set Inversion Solver

This solver, based on Set Inversion techniques and Modal Interval Analysis, allows solving problems involving quantified constraints over the reals through the characterization of their solution sets.

Quantified Contractor

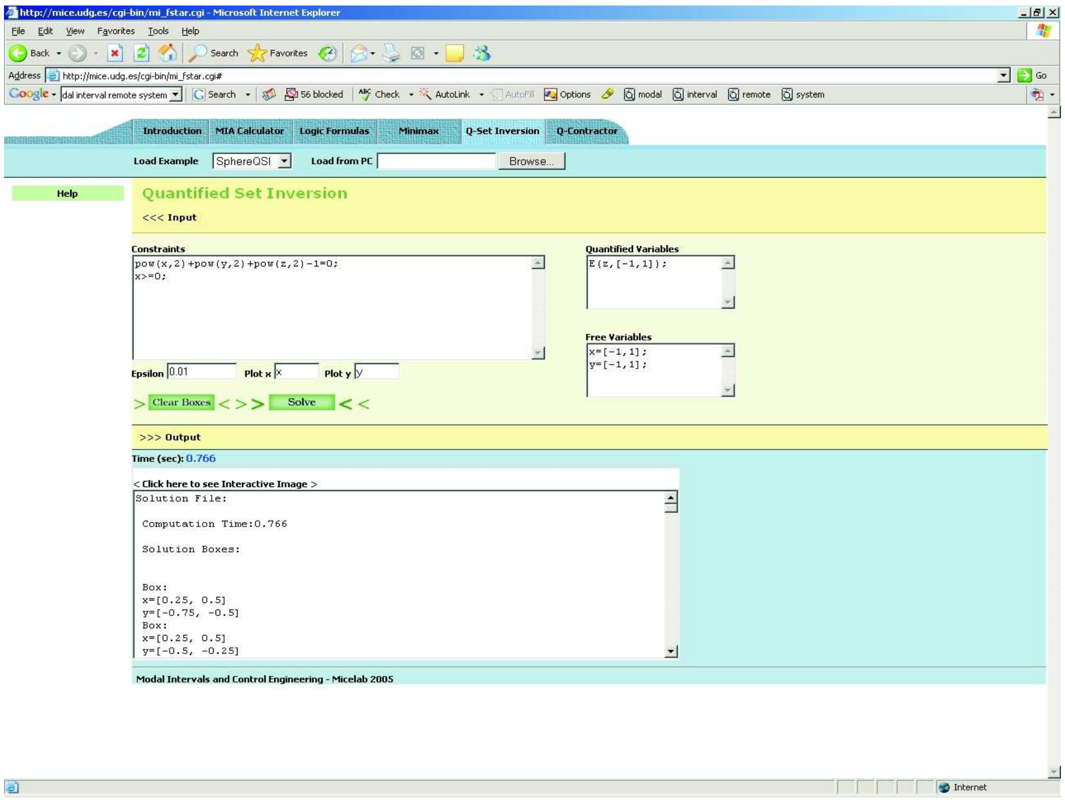
Given a system of nonlinear constraints over the real, which variables can be affected by logic quantifiers (Forall, Exists), this solver computes an outer interval estimation of its solution set.

Figure 8.15: Modal Interval Remote Solver main page.

3. Once the problem has been introduced, the user can submit it by clicking the Submit button. Then, the server recovers the data from the text boxes and generates a text file with the suitable syntax for the corresponding solver. For this purpose, a PERL parser has been employed.

4. The server executes the MISO with the generated text file as a parameter. The server accepts five simultaneous users. Each solver has a limited computation time in order to avoid too long computations.

5. The MISO generates a solution text file and the server processes it by means of a PERL parser. Then, it shows the results to the web page. The QSI solver provides a graphical output which is possible to visualize by clicking the link *Click here to see Interactive Image*.


http://mice.udg.es/cgi-bin/mi_fstar.cgi - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://mice.udg.es/cgi-bin/mi_fstar.cgi#
Introduction
MIA Calculator
Logic Formulas
Minimax
Q-Set Inversion
Q-Contractor
Help
Clear Boxes
Solve
Browse...
Internet

Load Example: SphereQSI
Load from PC

Quantified Set Inversion

<<< Input

Constraints
```
pow(x,2)+pow(y,2)+pow(z,2)-1=0;
x>=0;
```

Quantified Variables
```
E(z,[-1,1]);
```

Free Variables
```
x=[-1,1];
y=[-1,1];
```

Epsilon 0.01 Plot x x Plot y y

>>> Output

Time (sec): 0.766

< Click here to see Interactive Image >
```
Solution File:

 Computation Time:0.766

 Solution Boxes:


 Box:
 x=[0.25, 0.5]
 y=[-0.75, -0.5]
 Box:
 x=[0.25, 0.5]
 y=[-0.5, -0.25]
```

Modal Intervals and Control Engineering - Micelab 2005

Figure 8.16: MIRS interface corresponding to the QSI solver.

8.10 Conclusions

This chapter has overview the implementation of the different algorithms presented in this thesis. A high-level explanation has been given without entering into low-level details of the implementation. A web page allowing to remotely use the implemented algorithms in Internet, has also been presented. These solvers are still under development, therefore, they can contain some bugs and they are subject to modifications.

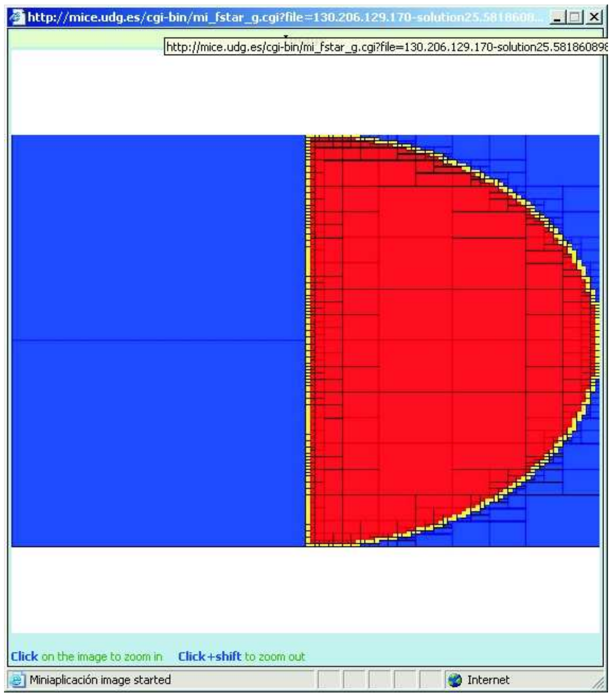

http://mice.udg.es/cgi-bin/mi_fstar_g.cgi?file=130.206.129.170-solution25.5818608...
http://mice.udg.es/cgi-bin/mi_fstar_g.cgi?file=130.206.129.170-solution25.58186089
Click on the image to zoom in
Click+shift to zoom out
Miniaplicación image started
Internet

Figure 8.17: MIRS graphical output for the QSI solver.

Chapter 9

Conclusions and Future Work

This chapter summarizes the principal contributions of this thesis. Suggestions for further work in some of the subjects encompassed by the research contained in this thesis are also included. At the end of the chapter, a list of related publications is provided.

9.1 Conclusions

This thesis has focused on the resolution of quantified real constraints (QRCs). In particular, a new methodology, based on Modal Interval Analysis (MIA), has been proposed for solving QRCs in an approximated, but guaranteed way.

Two main motivations have lead to the realization of this thesis. The first and most important motivation consists of making a contribution to the research being carried out on solving QRCs by improving some aspects of existing methods. The second motivation was the aim of exploiting the potential of MIA to solve problems that are considered to be difficult.

The methodology developed was validated by solving several problems from the literature, along with original problems. Comparisons with the state-of-the-art techniques on quantified real constraint solving were carried out, and the results obtained regarding the contribution of the proposed technique are convincing. The different algorithms presented in this thesis have been implemented in the form of solvers, which allows for easy use by a user.

The proposed methodology was also successfully applied to the resolution of real engineering problems formulated as QRCs. In the context of the European project, CHEM CHEM Consortium (2000), a software tool devoted to the detection of faults in dynamic systems was developed. This software tool, known as the SQUALTRACK solver, was applied to different industrial processes, and its usefulness was demonstrated. Another original application to the control of a sailboat was also implemented. This application combined the proposed methodology for solving QRCs with feedback linearization techniques. The viability of the proposed approach was demonstrated by means of simulation.

9.1.1 Contributions

The main contribution of this thesis can been seen from two different points of view. From the first viewpoint, this thesis exploits the full potential of MIA, something that has not been done before by previous works based on MIA. This was carried out by means of the resolution of QRCs, which are considered to be difficult problems. Moreover, this thesis identifies the main advantages and the principal drawbacks of using MIA approach with respect to the other existing approaches used for solving QRCs.

From the second viewpoint, this thesis represents a step forward in research into quantified real constraint solving. The proposed MIA-based approach, solves QRCs in a natural and elegant way. Moreover, using a comparison with some state-of-the-art techniques, it was proved that the proposed methodology was computationally more efficient in some cases, and it could solve problems that were out of the scope of current software implementations.

The more detailed contributions of this thesis are summarized as follows.

- Chapter 2 gave a precise problem definition and a detailed survey of the existing techniques used for solving QRCs, highlighting their advantages and limitations.

- Chapter 3 gave an intuitive introduction to MIA, providing several examples and a more standard mathematical notation. An efficient algorithm, f^* algorithm, used for the computation of modal interval extensions of a continuous function was presented. Different examples and comparisons with some state-of-the-art techniques were provided, which proved the efficiency of the proposed algorithm. An algorithm for proving the consistency of a class of QRCs was also presented. This algorithm was a direct application of the f^* algorithm to the resolution of QRCs.

- Chapter 4 gave a new algorithm for computing the inner an outer of the solution set for a class of QRCs. This algorithm was based on the f^* algorithm and branch-and-bound techniques, and is referred to as the "Quantified Set Inversion (QSI) algorithm". Several examples and a comparison with some state-of-the-art techniques showed the viability of the proposed approach.

- Chapter 5 gave an original continuous minimax optimization algorithm based on the f^* algorithm architecture. This work can be seen as a collateral result of the development of the f^* algorithm and represents an important and original application of MIA.

- Chapter 6 gave a formulation of the problem of robust fault detection in dynamic systems as a QRC. The implementation and validation in actual processes of a fault detection tool for dynamic systems was provided.

- Chapter 7 gave an original application of the presented QRC approach to the control of a sailboat. This combined the QSI algorithm with feedback linearization techniques. The approach was validated using simulation results.

- Chapter 8 provided the implementation of the presented algorithms in the thesis to facilitate and spread their utilization. A Web page that allowed for the online utilization of the algorithms was developed. All the solvers were provided, along with several examples and a corresponding user manual.

- Finally, the methodology developed in this thesis is currently being applied in other work. For example, work involving Computer Graphics Flórez *et al.* (2005), Structural Assessment Casas *et al.* (2005), and Fault Diagnosis Calderón-Espinoza *et al.* (2004) is using algorithms developed in this thesis.

9.2 Future work

This thesis represents a small contribution to the field of quantified real constraint solving. Therefore, future research is certainly needed. Some subjects where the research discussed in this thesis can continue are described in the following subsections.

9.2.1 Combining approaches

One of the main limitations of the existing approaches for solving QRCs is the computational complexity. Methods based on pruning techniques (e.g. Interval Constraint Propagation Benhamou & Older (1997)) try to reduce the exponential complexity of branch-and-bound algorithms by replacing expensive exhaustive searches using methods for pruning elements from the search space where it is easy to show that they do not contain solutions. However, in practice, it is observed that these techniques are inefficient when equality predicates are involved and the inner approximation of the solution set is required. In particular, the technique proposed in this thesis has been shown to be efficient in this case. Thus, an interesting area for future work consists of combining the present methodology with other existing approaches to take advantage of each of them.

9.2.2 Modal interval constraint propagation

Interval Constraint Propagation has been shown to be a successful approach for solving QRCs. However, it is limited with respect to the form of the QRCs that it can deal with, and there are many aspects concerning the computational complexity that can be improved on. An interesting line of research line is a study of the possible combination of Constraint Propagation techniques with

MIA. Some studies have already been carried out in that direction, but there are still some important theoretical aspects to be solved.

9.2.3 Solving the vectorial case

Solving QRCs with equality predicates sharing existentially quantified variables remains an open problem. Recent work by Goldsztejn Goldsztejn & Jaulin (2005) proposes an original method to tackle this problem. However, this approach is still not general enough, because it requires strong conditions in the form of involved constraints, and does not guarantee termination. Therefore, research into a more general approach for solving this problem should be considered for future work.

9.2.4 New applications

We envisage the possibility to apply the proposed approach for solving QRCs to *modal predictive control* and to *path planning and collision avoidance*. Finally, we are aware of recent application of MIA to mechanical design Wang (2006).

9.3 Related publications

The development of this thesis has generated a set of publications in different international journals, conferences, and workshops.

9.3.1 Publications in journals

- **Quantified Set Inversion algorithm with applications to control.**
  Pau Herrero, Miguel A. Sainz, Josep Vehi, Luc Jaulin.
  Reliable Computing, 2005, vol. 11-5, pag. 369 - 382.

- **Detección de fallos en procesos reales basada en modelos intervalares y múltiples ventanas temporales deslizantes.**
  Joaquim Armengol, Josep Vehí, Miguel Ángel Sainz, **Pau Herrero**.
  Computación y Sistemas, 2002, vol. 6-2, pag. 94 - 102.

- **Combining set computation and feedback linearization for control.**
  Pau Herrero, Luc Jaulin, Josep Vehí, Miguel A. Sainz.
  Automatica, Elsevier Science, Oxford, UK (Submitted).

- **An extended interval inclusion test for proving first-order logic formulas over the reals.**
  Miguel A. Sainz, **Pau Herrero**, Luc Jaulin, Joaquim Armengol.
  Journal of Applied Mathematics and Computation, Elsevier Science, Oxford, UK (Submitted).

- **Continuous minimax optimization using modal intervals.**
  Miguel A. Sainz, **Pau Herrero**, Josep Vehí, Joaquim Armengol.
  Journal of Mathematical Analysis and Applications, Elsevier Science, Oxford, UK (Submitted).

9.3.2 Publications in conferences

- **Parameter identification with quantifiers.**
  Remei Calm, Miguel A. Sainz, **Pau Herrero**, Josep Vehi, Joaquim Armengol.
  5th IFAC Symposiumon Robust Control Design (ROCOND06), 2006.

- **Detección de fallos en procesos reales (Proyecto CHEM).**
  Joaquim Armengol, **Pau Herrero**, Miguel Ángel Sainz.
  El análisis de intervalos en España: desarrollos, herramientas y aplicaciones, 2005, pag. 229 - 238.

- **Bridge nonitoring and assessment under uncertainty via Interval Analysis.**
  J. R. Casas, J. C. Matos, J.A. Figueiras, J. Vehí, O. García and **P. Herrero**.
  Proceedings of the 9th International Conference On Structural Safety And Reliability - ICOSSAR 2005 9th International Conference On Structural Safety And Reliability - ICOSSAR2005, pag. 487 - 494.

- **Inner and outer approximation of the polar diagram of a sailboat.**
  Pau Herrero, Luc Jaulin, Josep Vehí, Miguel A. Sainz.
  Interval analysis, constraint propagation, applications (IntCP05), 2005.

- **Visualization of Implicit Surfaces using Quantified Set Inversion.**
  Jorge Flórez, **Pau Herrero**, Miguel A. Sainz and Josep Vehi.
  Proceedings IntCP 2005 Interval Analysis, Constraint Propagation, Applications, 2005.

- **Combining interval and qualitative reasoning for fault diagnosis.**
  Gabriela Calderón-Espinoza, Joaquim Armengol, Miguel Ángel Sainz, **Pau Herrero**.
  Proceedings of IFAC World Congress 16th IFAC World Congress, 2005.

- **Application of interval models to the detection of faults in industrial processes.**
  Joaquim Armengol, Josep Vehí, Miguel Ángel Sainz, **Pau Herrero**.
  World Automation Congress 2004 (WAC 2004), 2004.

- **Industrial application of a fault detection tool based on interval models.**
  Joaquim Armengol, Josep Vehí, Miguel Ángel Sainz, **Pau Herrero**.
  International Conference on Integrated Modeling and Analysis in Applied Control and Automation 2004 (IMAACA 2004) (part of International Mediterranean Modeling Multiconference, I3M 2004), 2004.

- **Quantified Set Inversion with applications to control.**
  Pau Herrero, Miguel A. Sainz, Josep Vehí and Luc Jaulin.
  Proceedings 2004 IEEE CCA/ISIC/CACSD, 2004.

- **Fault detection in a pilot plant using interval models and multiple sliding time windows.**
  Joaquim Armengol, Josep Vehí, Miguel Ángel Sainz, **Pau Herrero**.
  Preprints SAFEPROCESS 2003 5th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS 2003), 2003, pag. 162 2002.

9.3.3 Publications in workshops

- **Solving problems on minimax optimization.**
  Miguel A. Sainz, **Pau Herrero**, Josep Vehi, Joaquim Armengol.
  PARA'04 Workshop on State-of-Art in Scientific Computing, 2004.

- **Quantified Set Inversion algorithm.**
  Pau Herrero, Miguel A. Sainz, Josep Vehí and Luc Jaulin.
  Proceedings 2nd International Workshop on Interval Mathematics and Constraint Programming IMCP-2004, 2004, pag. 272 - 279.

Appendix A

Problem Definitions

This appendix provides the sources for a set of problems solved along the thesis, to be introduced to the corresponding solvers presented in Chapter 8.

A.1 FSTAR Solver problems

Source 8 Program source for example 3.2.11.

```
% Parameters

Algorithm=FSTAR;

Epsilon=1e-4;

Tolerance=1e-4;

% Variables

u=[0,6];

v=[8,2];

z=[9,-4];

% Function

f:=pow(u,2)+pow(v,2)+2*u*v-20*u-20*v+100-10*sin(z);
```

A.2 QRCS Solver problems

Source 9 Program source for example 3.2.11.

```
% Parameters

Algorithm=QRCS;

Epsilon=1e-4;

Tolerance=1e-4;

% Variables

U(u,[0,6]);

E(v,[2,8]);

E(z,[-4,9]);

% Predicate

qc:=pow(u,2)+pow(v,2)+2*u*v-20*u-20*v+100-10*sin(z)=0;
```

Source 10 Program source for example 3.4.2.

```
% Parameters

Algorithm=QRCS;

Epsilon=0.0001;

Tolerance=0.0001;

% Variables

U(x1,[0.001,1]);

U(x2,[-0.3,-2]);

U(u,[-1,1]);

E(v,[-2,2]);

% Predicate

qc:=x1*u-x2*pow(v,2)*sin(x1)>0;
```

Source 11 Program source for example 3.4.3.

```
% Parameters

Algorithm=QRCS;

Epsilon=1e-1;

Tolerance=1e-1;

% Variables

U(x1,[-1,-0.5]);

U(x2,[-1,-0.5]);

E(u,[-0.5,0.5]);

% Predicate

qc:=min(-x1+x2*u+pow(x1,2),-x2+(1+pow(x1,2))*u+pow(u,3))>0;
```

A.3 QSI Solver problems

Source 12 Program source for example 4.2.1.

```
% Parameters

Algorithm=QSI;

Epsilon=1e-2;

Tolerance=1e-2;

CSPEps=0.05;

PlotX=x1;

PlotY=x2;

% Free Variables

F(x1,[-10,10]);

F(x2,[-10,10]);

% Quantified Variables

U(u,[-1,1]);

E(v,[-2,2]);

c:=x1*u-x2*pow(v,2)*sin(x1)>0;
```

Source 13 Program source for example 4.2.2.

```
% Parameters

Algorithm=QSI;

Epsilon=1e-2;

Tolerance=1e-2;

CSPEps=0.05;

PlotX=x1;

PlotY=x2;

% Free Variables

F(x1,[-1,1]);

F(x2,[-1,1]);

E(u,[-0.5,0.5]);

% Constraints

c:=min(-x1+x2*u+pow(x1,2),-x2+(1+pow(x1,2))*u+pow(u,3))>0;
```

Source 14 Program source for example 1 of Section 4.3.1.

```
Algorithm=QSI;

Epsilon=1e-3;

Tolerance=1e-3;

CSPEps=0.01;

PlotX=b;

PlotY=c;

% Variables

F(q,[-3,3]);

U(p1,[0, 1]);

% Constraints

c:=9+48*p1+48*q+32*p1*q>=0;

c:=1+p1+q>=0;

c:=-16*p1-16*q+16*pow(p1,2)+16*pow(q,2)+7>=0;
```

Source 15 Program source for example 2 of Section 4.3.1.

```
% Parameters

Algorithm=QSI;

Epsilon=1e-2;

Tolerance=1e-2;

CSPEps=0.1;

PlotX=c1;

PlotY=c2;

% Free variables

F(c1,[0.1,1]);

F(c2,[0.1,1]);

% Quantified variables

U(p1,[0.9,1.1]);

U(p2,[0.9,1.1]);

U(p3,[0.9,1.1]);

% Constraints

c:=(1+c2*p1)*((p2*pow(p3,2)+p3)*(p2*p3+1)-p2*(pow(p3,2)+

c2*p1*pow(p3,2)))-pow(p2*p3+1,2)*(c1*p1) > 0;

c:=pow(p2*p3+1,2) - p2*(p3+c2*p1*p3) > 0;

c:=c1*p1*pow(p3,2) > 0;
```

Source 16 Program source for example of Section 4.3.2.

```
% Parameters

Algorithm=QSI;

Epsilon=1e-2;

Tolerance=1e-2;

CSPEps=0.01;

PlotX=x1;

PlotY=x2;

% Variables

F(x1,[-2,2]);

F(x2,[-2,2]);

E(x3,[-1,1]);

% Constraints

c:=pow(x1,2)+pow(x2,2)-x3=0;
```

Source 17 Program source for example of Section 4.3.3.

```
% Parameters
Algorithm=QSI;
Epsilon=1e-2;
Tolerance=1e-2;
CSPEps=0.01;
PlotX=p1;
PlotY=p2;
% Free variables
F(p1,[0,1.2]);
F(p2,[0,0.5]);
%Quantified variables
E(t1,[-0.25,1.75]);
E(t2,[0.5,2.5]);
E(t3,[1.25,3.25]);
E(t4,[2,4]);
E(t5,[5,7]);
E(t6,[8,10]);
E(t7,[12,14]);
E(t8,[16,18]);
E(t9,[20,22]);
E(t10,[24,26]);
E(c1,[2.7,12.1]);
E(c2,[1.04,7.14]);
E(c3,[-0.13,3.61]);
E(c4,[-0.95,1.15]);
E(c5,[-4.85,-0.29]);
E(c6,[-5.06,-0.36]);
E(c7,[-4.1,-0.04]);
E(c8,[-3.16,0.3]);
E(c9,[-2.5,0.51]);
E(c10,[-2,0.67]);
%Constraints
c:=20*exp(-p1*t1)-8*exp(-p2*t1)-c1=0;
c:=20*exp(-p1*t2)-8*exp(-p2*t2)-c2=0;
c:=20*exp(-p1*t3)-8*exp(-p2*t3)-c3=0;
c:=20*exp(-p1*t4)-8*exp(-p2*t4)-c4=0;
c:=20*exp(-p1*t5)-8*exp(-p2*t5)-c5=0;
c:=20*exp(-p1*t6)-8*exp(-p2*t6)-c6=0;
c:=20*exp(-p1*t7)-8*exp(-p2*t7)-c7=0;
c:=20*exp(-p1*t8)-8*exp(-p2*t8)-c8=0;
c:=20*exp(-p1*t9)-8*exp(-p2*t9)-c9=0;
c:=20*exp(-p1*t10)-8*exp(-p2*t10)-c10=0;
```

Source 18 Program source for example of Section 4.3.4.

```
% Parameters

Algorithm=QSI;

Epsilon=1e-2;

Tolerance=1e-2;

CSPEps=0.1;

PlotX=x;

PlotY=y;

% Free variables

F(x,[0,1]);

F(y,[-1,1]);

% Quantified variables

E(u3,[-1,1]);

% Constraints

c:=139*y-112*x*y-388*pow(x,2)*y+215*pow(x,3)*y-38*pow(y,3)+

185*x*pow(y,3)-(-38*y-170*x*y+148*pow(x,2)*y+4*pow(y,3)+

u3*(14-10*x+37*pow(x,2)-48*pow(x,3)+8*pow(x,4)-13*pow(y,2)-

13*x*pow(y,2)+20*pow(x,2)*pow(y,2)+11*pow(y,4)))/

(-52-2*x+114*pow(x,2)-79*pow(x,3)+7*pow(y,2)+14*x*pow(y,2))

*(-11+35*x-22*pow(x,2)+5*pow(x,2)+10*pow(x,3)-17*x*pow(y,2))+

u3*(-44+3*x-63*pow(x,2)+34*pow(y,2)+142*pow(x,3)+63*x*pow(y,2)-

54*pow(x,4)-69*pow(x,2)*pow(y,2)-26*pow(y,4))=0;
```

A.4 MINIMAX Solver problems

Source 19 Program source for example 5.3.1.

```
% Parameters

Algorithm=MINIMAX;

Epsilon=1e-6;

Tolerance=1e-6;

Optim=1;

Solution=1;

% Variables

MIN(x1,[0,6]);

MAX(x2,[2,8]);

% Function

f:=pow(x1,2)+pow(x2,2)+2*x1*x2-20*x1-20*x2+100;
```

Source 20 Program source for example 5.3.2.

```
% Parameters

Algorithm=MINIMAX;

Epsilon=1e-3;

Tolerance=1e-3;

Optim=1;

Solution=1;

% Variables

MIN(x,[-3.14, 3.14]);

MAX(y,[-3.14, 3.14]);

% Function

f:=pow(cos(y)+cos(2*y+x),2);
```

Source 21 Program source for example 5.3.3.

```
% Parameters

Algorithm=MINIMAX;

Epsilon=1e-3;

Tolerance=1e-2;

Optim=1;

Solution=1;

% Variables

MIN(x1,[-1, 2]);

MAX(x2,[-1, 1]);

MAX(x3,[-1, 1]);

% Function

f:=pow(exp(-0.1*x1)-exp(-0.1*x2)-(exp(-0.1)-exp(-1))*x3,2)+

pow(exp(-0.2*x1)-exp(-0.2*x2)-(exp(-0.2)-exp(-2))*x3,2)+

pow(exp(-0.3*x1)-exp(-0.3*x2)-(exp(-0.3)-exp(-3))*x3,2)+

pow(exp(-0.4*x1)-exp(-0.4*x2)-(exp(-0.4)-exp(-4))*x3,2)+

pow(exp(-0.5*x1)-exp(-0.5*x2)-(exp(-0.5)-exp(-5))*x3,2)+

pow(exp(-0.6*x1)-exp(-0.6*x2)-(exp(-0.6)-exp(-6))*x3,2)+

pow(exp(-0.7*x1)-exp(-0.7*x2)-(exp(-0.7)-exp(-7))*x3,2)+

pow(exp(-0.8*x1)-exp(-0.8*x2)-(exp(-0.8)-exp(-8))*x3,2)+

pow(exp(-0.9*x1)-exp(-0.9*x2)-(exp(-0.9)-exp(-9))*x3,2)+

pow(exp(-x1)-exp(-x2)-(exp(-1)-exp(-10))*x3,2);
```

Source 22 Program source for example 5.3.5.

```
% Parameters

Algorithm=MINIMAX;

Epsilon=1e-3;

Tolerance=1e-3;

Optim=1;

Solution=1;

% Variables

MIN(x1,[0,6]);

MAX(x2,[2,8]);

% Function

f:=pow(x1,2)+pow(x2,2)+2*x1*x2-20*x1-20*x2+100;

% Constraints

c:=pow(x1-5,2)+pow(x2-3,2)-4>0;

c:=pow(x1-5,2)+pow(x2-3,2)-16<0;
```

Source 23 Program source for example 5.3.6.

```
% Parameters

Algorithm=MINIMAX;

Epsilon=1e-3;

Tolerance=1e-3;

Optim=1;

Solution=1;

% Variables

MIN(x1,[0,6]);

MAX(x2,[2,8]);

% Function

MIN(x,[-3.14, 3.14]);

MAX(y,[-3.14, 3.14]);

f:=pow(cos(y)+cos(2*y+x),2);

% Constraints

c:=y-x*(x+2*3.14)<0;

c:=y-x*(x-2*3.14)<0;
```

A.5 SQUALTRACK Solver problems

Source 24 Program source for example of Section 6.3.1.

```
Algorithm=SQT;

%Parameters

InFile=reactor_leakage;

OutFile=Solution;

SampleTime=3;

Windows={5,50,75,100};

%Measurements

Measurement(F1,0.03,0);

Measurement(F3,0.03,0);

%State variable

State(LT1,0.05,0);

%Function model

f:=LT1+((F3+20)/3600-F1/3600);
```

Source 25 Program source for example of Section 6.3.2.

```
Algorithm=SQT;

%Parameters

InFile=resistor_shutdown;

OutFile=Solution;

SampleTime=3;

WindowGraph=1000;

Windows={10,25};

%Measurements

Measurement(LT1,0.03,0);

Measurement(F3,0.05,0);

Measurement(R1,0,0);

%Model parameters

Parameter(Loss,[0.01,0.02]);

Parameter(Ts,[3,3]);

Parameter(Tenv,[18,19]);

Parameter(Ro,[999,1001]);

Parameter(Ce,[4185,4187]);

%State variable

State(T1,0.03,0);

%Function model

f:=(1-Ts/(LT1/1000)*(F3/3600000+Loss/(Ro*Ce)))*T1+(Ts/(LT1/1000))*

(F3*[18,20]/3600000+(R1*[24,30]-Loss*Tenv)/(Ro*Ce));
```

Source 26 Program source for example of Section 6.3.3.

```
Algorithm=SQT;

%Parameters

InFile=esc31;

OutFile=Solution;

SampleTime=3;

Windows={2,5,10,15};

%Model parameters

Parameter(A,[0.97,0.999]);

Parameter(B,[0.00095,0.0315]);

%Measurements

Measurement(U,0.05,0.001);

%State variable

State(Y,0.05,0);

%Function model

f:=A*Y+B*U;
```

References

Abdallah, C., Dorato, P., Yang, W., Liska, R. & Steinberg, S. (1996). Applications of quantifier elimination theory to control system design. *4th IEEE Mediterranean Symposium on Control and Automation, Crete, Greece.*. 1

Abdallah, C., Dorato, P., Famularo, D. & Yang, W. (1999). Robust nonlinear feedback design via quantifier elimination theory. *International Journal of Robust and Non-Linear Control*, **9**, 817–822. 16

Armengol, J. (1999). *Application of Modal Interval Analysis to the Simulation of The Behaviour of Dynamic Systems with Uncertain Parameters*. Ph.D. thesis, Universitat de Girona. Catalonia, Spain. 4, 114

Armengol, J., Vehí, J., Sainz, M.Á. & Herrero, P. (2003). Fault detection in a pilot plant using interval models and multiple sliding time windows. In *5th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2003. Washington, D.C., U.S.A.*. 114, 121

Armengol, J., Vehí, J., Sainz, M.Á. & Herrero, P. (2004). Industrial application of a fault detection tool based on interval models. In *International Conference on Integrated Modeling and Analysis in Applied Control and Automation 2004 (IMAACA 2004) (part of International Mediterranean Modeling Multiconference, I3M 2004)*. 114, 121

Balakrishnan, K. & Honavar, V. (1998). Intelligent diagnosis systems. *Journal of Intelligent Systems*, **8 (3)**, 239–290. 114

Basso, P. (1985). Optimal search for the global maximum of functions with bounded seminorm. *SIAM J. Numer. Anal.*, **9**, 888–903. 95

Benhamou, F. & Goualard, F. (2000). Universally quantified interval constraints. 5, 8, 29

Benhamou, F. & Older, W. (1997). Applying interval arithmetic to real, integer, and boolean constraints. *Journal of Logic Programming*, **32**, 1–24. 2, 12, 16, 118, 191

Benhamou, F., Goualard, F., Granvilliers, L. & Puget, J.F. (1999). Revising hull and box consistency. In *Proc. Sixteenth Int. Conf. on Logic Programming (ICLP'99)*, the MIT Press. 17, 19, 21

Benhamou, F., Goualard, F., Languenou, E. & Christie, M. (2004). Interval constraint solving for camera control and motion planning. *ACM Trans. Comput. Logic*, **5**, 732–767. 29

Blanke, M., Kinnaert, M., Lunze, J. & Staroswiecki, M. (2003). Diagnosis and fault-tolerant control. *Springer*. 115

BOOST (2006). *Boost C++ Libraries*. http://www.boost.org/. 156

Bordeaux, L. & Monfroy, E. (2002). Beyond np: Arc-consistency for quantified constraints. In *Proceedings CP-2002*, Ithaca, NY. 8

Bordeaux, L., Monfroy, E. & Benhamou, F. (2001). Improved bounds on the complexity of kb-consistency. *Proceedings of 17th Int. Joint Conf. on Artificial Intelligence*. 25

Brown, C. (2004). *Quantifier Elimination by Partial Cylindrical Algebraic Decomposition*. http://www.cs.usna.edu/~qepcad/B/QEPCAD.html. 15

Builder, C. (2006). *Borland C++ Builder*. http://www.borland.com/us/products/cbuilder/index.html. 180

Calderón-Espinoza, G., Armengol, J., Sainz, M.Á. & Herrero, P. (2004). Combining interval and qualitative reasoning for fault diagnosis. In *Proceedings of the 16th IFAC World Congress*. 135, 191

Calm, R. (2006). *Análisis intervalar modal: su construcción teórica, implementación y posibilidades de aplicación a la simulación y al control*. Ph.D. thesis, Universitat de Girona. Catalonia, Spain. 4

Calm, R., Sainz, M., Herrero, P., Vehi, J. & Armengol, J. (2006). Parameter identification with quantifiers. *5th IFAC Symposiumon Robust Control Design (ROCOND06)*. 72

Casas, J.R., Matos, J.C., Figueiras, J.A., Vehí, J., García, O. & Herrero, P. (2005). Bridge nonitoring and assessment under uncertainty via interval analysis. *Proceedings of the 9th International Conference On Structural Safety And Reliability - ICOSSAR 2005 9th*. 191

CCA2006 (2006). *Computability and Complexity in Analysis*. http://cca-net.de/cca2006/. 66

(CGI) (2006). *Common Gateway Interface*. http://hoohoo.ncsa.uiuc.edu/cgi/. 183

Chauvin, C. & Müller, M. (1951). An application of quantifier elimination to mathematical biology. 16

Chauvin, C., Muller, M. & Weber, A. (1994). An application of quantifier elimination to mathematical biology. *In Computer Algebra in Science and Engineering.World Scientific*, 287–296. 2

CHEM Consortium (2000). Advanced decision support system for chemical/petrochemical manufacturing processes. 114, 121, 122, 174, 180, 189

Code::Blocks (2006). *The open source, cross platform Free C++ IDE.*. http://www.codeblocks.org/. 154

Collavizza, H., Delobel, F. & Rueher, M. (1999). Comparing partial consistencies. *Reliable Computing*, **5**, 1–16. 25

Collins, G.E. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *2nd GI Conf. Automata Theory and Formal Languages*, vol. 33, 134–189, Springer. 8, 14, 26

Dao, M. (2005). *Proj2D Solver*. http://www.istia.univ-angers.fr/∼dao/. 29, 79, 88, 144

Dao, M., Baguenard, X. & Jaulin, L. (2003). Projection d'ensembles pour l'estimation de parameters, la conception de robot et la commande robuste. *Journées Doctorale d'Automatique - Valenciennes*. 29

Davenport, J.H. & Heintz, J. (1988). Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, **5**, 29–35. 2, 14

Davis, E. (1975). Understanding line drawings of scenes with shadows. *In Winston, P. H., editor, The Psychology of Computer Vision. McGraw-Hill, New York.*. 16

de Guzman, J. (2006). *Spirit Framework*. http://spirit.sourceforge.net/. 155

Demyanov, V.F. & Malozemov, V.N. (1990). Introduction to minimax. *Dover Publications Inc., New York USA*. 96, 106, 112

Dorato, P. (2000). Quantified multivariate polynomial inequalities. *IEEE Control Systems Magazine*, **20(5)**, 48–58. 1, 78

Dorato, P., Yang, P. & Abdallah, C. (1997). Robust multi-objective feedback design by quantifier elimination. *J. Symbolic Computation*, **24(2)**, 153–159–58. 16

Ebbinghaus, H.D., Flum, J. & Thomas, W. (1984). Mathematical logic. *Springer Verlang*. 8, 9

EBNF (2003). *Extended Backus Normal Form*. http://www.garshol.priv.no/download/text/bnf.html/. 156

Eckel, B. (2000). *Thinking in C++*. PlanetPDF.com. 154

Elkaim, G. & Kelbley, R. (2006). Control architecture for segmented trajectory following of a wind-propelled autonomous catamaran. *Proceedings of the AIAA Guidance, Navigation, and Control Conference, AIAA GNC2006, Keystone, CO*. 137

Elkaim, G., Woodley, B. & Kelbley, R. (2006). Model free subspace h-infinity control for an autonomous catamaran. *Proceedings of the ION/IEEE Position, Location, and Navigation Symposium, ION/IEEE PLANS2006, San Diego, CA*. 137

Fliess, M., Lévine, J., Martin, P. & Rouchon, P. (1995). Flatness and defect of non-linear systems: Introductory theory and examples. *International Journal of Control*, **61(6)**, 1327–1361. 137, 145

Flórez, J., Herrero, P., Sainz, M.A. & Vehi, J. (2005). Visualization of implicit surfaces using quantified set inversion. *Proceedings IntCP 2005*. 72, 84, 85, 190

FSF (2006). *The Free Software Foundation*. http://www.fsf.org/. 155

García Reyero, S. & Martínez, J.L. (1999). Modal intervalar arithmetic implementation using floating point emulation. *Workshop on Applications of Interval Analysis to Systems and Control with special emphasis on recent advances in Modal Interval Analysis (MISC 99). Girona, Catalonia, Spain*, 211–223. 159

Gardeñes, E., Sainz, M.Á., Jorba, L., Calm, R., Estela, R., Mielgo, H. & Trepat, A. (2001). Modal intervals. *Reliable Computing*, **7**, 77–111. x, 4, 28, 33, 35, 43, 97

Garloff, J. (1985). Convergent bounds for the range of multivariate polynomials. *Interval Mathematics 1985*, 37–56. 12, 16

Garloff, J. (1993). The bernstein algorithm. *Interval Computations*, **2**, 154–168. 5, 16, 27

GCC (2006). *GCC compiler*. http://gcc.gnu.org/. 155

Gent, I. & Walsh, T. (1999). Beyong NP: The QSAT phase transition. In *Proceedings of ECAI*, 599–603. 8

GNU (2006). *GNU enviorenment*. http://www.gnu.org/. 155

Goldsztejn, A. (2003). Verified projection of the solution set of parametric real systems. *Proc of 2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS'03), Lausanne, Switzerland.*. 28

Goldsztejn, A. (2005). A right-preconditioning process for the formal-algebraic approach to inner and outer estimation of AE-solution set. *Reliable Computing*, **11(6)**, 443–478. 28

Goldsztejn, A. & Jaulin, L. (2005). Inner approximation of the range of vector-valued functions. *Reliable Computing (Submitted for publication).* 29, 192

Goldsztejn, A., Daney, D., Rueher, M. & Taillibert, P. (2005). Modal intervals revisited: A mean-value extension to generalized intervals. *First International Workshop on Quantification in Constraint Programming, Sitges, Spain, 2005 (QCP 2005).* 50

Hansen, E. (1992). *Global Optimization Using Interval Analysis*. Marcel Dekker. 4, 66, 95, 107

Hansen, E. & Sengupta, S. (1981). Bounding solutions of systems of equations using interval analysis. *BIT*, **21**, 203211. 29

Herrero, P. & Jaulin, L. (2006). *Modal Control of a SailBoat*. http://eia.udg.es/∼pherrero/software.htm. 151

Herrero, P., Jaulin, L., Sainz, M. & Vehi, J. (2004a). Quantified set inversion algorithm. *Proceedings 2nd International Workshop on Interval Mathematics and Constraint Programming IMCP-2004*, 272 – 279. 72

Herrero, P., Jaulin, L., Sainz, M. & Vehi, J. (2004b). Quantified set inversion with applications to control. *Proceedings 2004 IEEE CCA/ISIC/CACSD*. 72

Herrero, P., Jaulin, L., Sainz, M. & Vehi, J. (2005a). Inner and outer approximation of the polar diagram of a sailboat. *Interval analysis, constraint propagation, applications (IntCP05)*. 72, 137

Herrero, P., Vehí, J., Sainz, M.Á. & Jaulin, L. (2005b). Quantified set inversion algorithm. *Reliable Computing*, **11**, 369 – 382. 72

Herrero, P., Jaulin, L., Sainz, M. & Vehi, J. (2006). Sailboat control using set computation and feedback linearization. *Automatica (submited)*. 137

Hong, H. (1990). Improvements in CAD-based quantifier elimination. *PhD thesis - Ohio State University, Columbus, Ohio*. 15

Hong, H. (1993). Quantifier elimination for formulas constrained by quadratic equations via slope resultants. *The Computer Journal*, **36**, 440–449. 16

Hong, H. (1995). Symbolic-numeric methods for quantified constraint solving. *International Symposium on Scientific Computating, Computed Arithmetic and Validated Numerics SCAN-95*. 26

Horowitz, E., Sahni, S. & Rajasekaran, S. (1997). *Computer Algorithms - C++*. W. H. Freeman Press. 25, 66

HTML (2006). *HyperText Markup Language*. http://www.w3.org/MarkUp/. 183

Ioakimidis, N.I. (1999). REDLOG-aided derivation of feasibility conditions in applied mechanics and engineering problems under simple inequality constraints. *Journal of Mechanical Engineering (Strojnícky Casopis)*, **50(1)**, 58–69. 2

Jaulin, L. (2001a). Reliable minimax parameter estimation. *Applied Mathematics and Computation*, **7(3)**, 231–246. 95

JAULIN, L. (2001b). *Représentation d'État Pour la Modélisation et la Command des Systèmes*. Hermes. 140

JAULIN, L. & WALTER, E. (1993). Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, **32(8)**, 1053–1064. 5, 27, 28, 71, 77

JAULIN, L. & WALTER, É. (1996). Guaranteed tuning, with application to robust control and motion planning. *Automatica*, **32(8)**, 1217–1221. 1, 27

JAULIN, L. & WALTER, E. (1999). Guaranteed bounded-error parameter estimation for nonlinear models with uncertain experimental factors. *Automatica*, **35**, 849–856. 27, 28, 85, 88

JAULIN, L., KIEFFER, M., DIDRIT, O. & WALTER, É. (2001). *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer, London. 4, 95

JAULIN, L., BRAEMS, I. & WALTER, E. (2002). Interval methods for nonlinear identification and robust control. *CDC2002 - Conference on Decision and Control - La Vegas*. 5, 29, 80, 81

JIRSTRAND, M. (1997). Nonlinear control system design by quantifier elimination. *Journal of Symbolic Computation*, **24(2)**, 137–152. 1, 2, 16

KAHAN, W. (1996). Lecture notes on the status of IEEE standard 754 for binary floating point arithmetic. 45, 46, 160

KAUCHER, E.W. (1980). Interval analysis in the extended interval space IR. *Computing (Suppl.)*, **2**, 3349. 28, 38

KEARFOTT, R.B. (2001). Interval analysis: Interval fixed point theory. *Optimization*, **3**, 4851. 28

LEX-YACC (2006). *Lex and Yacc*. http://dinosaur.compilertools.net/. 155

LHOMME, O. (1993). Consistency techniques for numerical CSPs. *In International Joint Conference on Artificial Intelligence (IJCAI)*, **1**, 232–238. 21

MacCallum, S. (1988). Solving polynomial strict inequalities using cylindrical algebraic decomposition. *The Computer Journal*, **36**, 432–438. 16

Makino, K. & Berz, M. (2003). Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, **4**, 379–456. 16

Malan, S., Milanese, M. & Taragna, M. (1992). b^3 algorithm for robust performance analysis in presence of mixed parametric and dynamic perturbations. *Proc. 31st IEEE Conf. Decisison and Control, Tucston, AZ.*, 128–133. 27

Malan, S., Milanese, M. & Taragna, M. (1997). Robust analysis and design of control systems using interval arithmetic. *Automatica*, **33(7)**, 1363–1372. 27

MiceLab (2005). *Fstar Remote System*. http://mice.udg.es/fstar. 184

Microsystems., S. (1996). FDLIBM: Freely distributable LIBM. 160

Microtransat (2006). *Microtransat Challenge Cup*. http://www.ensica.fr/microtransat. 136

MONET (2006). *European network of excellence on model based systems and qualitative reasoning*. http://monet.aber.ac.uk:8080/monet/index.html. 115

Moore, R.E. (1966). *Interval Analysis*. Prentice-Hall. 2, 3, 12, 16, 17, 26, 32, 118

Neumaier, A. (1990). *Interval Methods for Systems of Equations*. Unv. Press, Cambridge. 3, 28

Nickel, K. (1986). Optimization using interval mathematics. 95

Patton, R.J., Frank, P.M. & Clark, R.N. (2000). Issues of fault diagnosis for dynamic systems. *Springer*. 115

Perl (2006). *Perl Language*. http://www.perl.com/. 183

Ratschan, S. (2001). Quantified constraints under perturbation. *Journal of Symbolic Computation*, **33**, 493–505. 25, 26

Ratschan, S. (2002a). *Approximate Quantified Constraint Solving*. http://www.mpi-sb.mpg.de/∼ratschan/AQCS/AQCS.html. 26, 81, 144

Ratschan, S. (2002b). Approximate quantified constraint solving by cylindrical box decomposition. *Reliable Computing*, **8**, 21–42. 26

Ratschan, S. (2002c). Continuous first-order constraint satisfaction with equality and disequality constraints. In P. van Hentenryck, ed., *Proc. 8th International Conference on Principles and Practice of Constraint Programming*, no. 2470 in LNCS, 680–685, Springer. 29

Ratschan, S. (2003a). Convergent approximate solving of first-order constraints by approximate quantifiers. *ACM Transactions on Computational Logic*. 26

Ratschan, S. (2003b). Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*. 5, 29

Ratschan, S. (2003c). Solving existentially quantified constraints with one equality and arbitrarily many inequalities. In F. Rossi, ed., *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*, no. 2833 in LNCS, 615–633, Springer. 8

Ratschan, S. (2005). *RSOLVER*. http://rsolver.sourceforge.net/. 29, 67, 79, 81, 88, 144

Sailbot (2006). *Autonomous Sailboat Competition*. http://engsoc.queensu.ca/sailboat/competition/index.htm. 137

Sainz, M., Gardeñes, E. & Jorba, L. (2002a). Formal solutions to systems of linear or non-linear equations. *Reliable Computing*, **8**, 189–211. 28

Sainz, M., Gardeñes, E. & Jorba, L. (2002b). Interval estimations of solution sets to real-valued systems of linear or non-linear equations. *Reliable Computing*, **8**, 283–305. 28

Sainz, M., Herrero, P., Vehi, J. & Armengol, J. (2004). Solving problems on minimax optimization. *PARA'04 Workshop on State-of-Art in Scientific Computing*. 98

Sainz, M., Herrero, P., Vehi, J. & Armengol, J. (2006a). Continuous minimax optimization using modal intervals. *Journal of Mathematical Analysis and Applications, Elsevier Science, Oxford, UK (Submitted)*. 98

Sainz, M., Herrero, P., Vehi, J. & Armengol, J. (2006b). An extended interval inclusion test for proving first-order logic formulas over the reals. *Journal of Applied Mathematics and Computation, Elsevier Science, Oxford, UK (Submitted)*. 51

Sainz, M.Á., Armengol, J., Vehí, J. & Herrero, P. (2002c). Detección de fallos en procesos reales basada en modelos intervalares y múltiples ventanas temporales deslizantes. *Computación y Systemas*, **6**, 94–102. 114

SamHaroud, D. (1995). Constraint consistency techniques for continuous domains. *PhD dissertation 1423, Swiss Federal Institude of Technology in Lausanne, Switzerland.*. 2, 16

Shary, S.P. (2002). A new technique in systems analysis under interval uncertainty and ambiguity. *Reliable Computing*, **8**, 321–418. 28

SIGLA/X (1999). *Applications of Interval Analysis to Systems and Control*, chap. Modal Intervals, 157–227. Universitat de Girona. 35

Sotiropoulos, D.G. (2004). Solving discrete minimax problems using interval arithmetic. *International Conference on Optimization: Techniques and Applications (ICOTA)*. 95

Stancu, V., A. Puig & Quevedo, J. (2005). Model-based robust fault detection using a forward backward test. *Interval Analysis and Constraint Propagation for Applications IntCP 2003*. 118

Stevens, B.L. & Lewis, F. (1993). Aircraft control and simulation. *Wiley*. 90

STL (2006). *Standard Template Library*. http://www.sgi.com/tech/stl/. 155

Sturm, T. (2000). Reasoning over networks by symbolic methods. *Applicable Algebra in Engineering Communication and Computing*, **10(1)**, 79–96. 2

Tarski, A. (1951). A decision method for elementary algebra and geometry. *Univ. of California Press, Berkeley*. 2, 13

Trepat, A. (1982). Completacion reticular del espacio de intervalos. *Tesina de Licenciatura*. 51

Vehí, J. (1998). *Anàlisi i Disseny de Controladors Robustos Mitjançant Intervals Modals*. Ph.D. thesis, Universitat de Girona. Catalonia, Spain. 4, 28

Vehí, J., Rodellar, J., Sainz, M.Á. & Armengol, J. (1999). Necessary and sufficient conditions for robust stability using modal intervals. *42nd IEEE Midwest Symposium on Circuits and Systems*, **2**, 673 – 676. 27

Vehí, J., Rodellar, J., Sainz, M.Á. & Armengol, J. (2000). Analysis of the robustness of predictive controllers via modal intervals. *Reliable Computing*, **6**, 281–301. 27

Venkatasubramanian, V., Rengaswamy, R., Yin, K. & Kavuri, S.N. (2003). IA review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Computers and Chemical Engineering*, **27**, 293–311. 114

Vicino, A., Tesi, A. & Milanese, M. (1990). Computation of nonconservative stability perturbation bounds for systems with nonlineary correlated uncertainty. *IEEE Trans. Automat. Contr.*, **35**, 835–841. 27

Wang, Y. (2006). Semantic tolerance modeling based on modal interval analysis. *Proceedings of NSF Workshop on Reliable Engineering Computing (REC'06)*. 192

wxWidgets (2006). *A cross-platform GUI library*. http://www.wxwidgets.org/. 158

Zettler, M. & Garloff, J. (1998). Robustness analysis of polynomial parameter dependency using bernstein expansion. *IEEE Trans. Automat. Contr.*, **43**, 425–431. 27

Zuche, S., Huang, Z. & Wolfe., M.A. (1997). An interval maximum entropy method for a discrete minimax problem. *Applied Mathematics and Computation*, **87**, 49–68. 95

Zuhe, S., Neumaier, A. & Eiermann, M. (1990). Solving minimax problems by interval methods. *BIT*, **30**, 742–751. 96