Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex

Grant Olney Passmore

Doctor of Philosophy

Mathematical Reasoning Group

Algorithms and Complexity Group

LFCS, School of Informatics

University of Edinburgh

2011

Abstract

We describe contributions to algorithmic proof techniques for deciding the satisfiability of boolean combinations of many-variable nonlinear polynomial equations and inequalities over the real and complex numbers.

In the first half, we present an abstract theory of Gröbner basis construction algorithms for algebraically closed fields of characteristic zero and use it to introduce and prove the correctness of Gröbner basis methods tailored to the needs of modern satisfiability modulo theories (SMT) solvers. In the process, we use the technique of proof orders to derive a generalisation of S-polynomial superfluousness in terms of transfinite induction along an ordinal parameterised by a monomial order. We use this generalisation to prove the abstract ("strategy-independent") admissibility of a number of superfluous S-polynomial criteria important for efficient basis construction. Finally, we consider local notions of proof minimality for weak Nullstellensatz proofs and give ideal-theoretic methods for computing complex "unsatisfiable cores" which contribute to efficient SMT solving in the context of nonlinear complex arithmetic.

In the second half, we consider the problem of effectively combining a heterogeneous collection of decision techniques for fragments of the existential theory of real closed fields. We propose and investigate a number of novel combined decision methods and implement them in our proof tool **RAHD** (Real Algebra in High Dimensions). We build a hierarchy of increasingly powerful combined decision methods, culminating in a generalisation of partial cylindrical algebraic decomposition (CAD) which we call Abstract Partial CAD. This generalisation incorporates the use of arbitrary sound but possibly incomplete proof procedures for the existential theory of real closed fields as first-class functional parameters for "short-circuiting" expensive computations during the lifting phase of CAD. Identifying these proof procedure parameters formally with **RAHD** proof strategies, we implement the method in **RAHD** for the case of full-dimensional cell decompositions and investigate its efficacy with respect to the Brown-McCallum projection operator.

We end with some wishes for the future.

Acknowledgements

Pursuing this work has been for me some kind of paradise. There are so many whom I wish to thank.

First and foremost, I thank my PhD supervisor, Paul B. Jackson. Paul's encouragement, direction, intellectual dexterity and endless positivity have made this dissertation a joy to compose. I find it hard to imagine how one could have a better PhD supervisor than mine. Paul was ever accessible, always found time to assist me with unexpected difficulties (technical or otherwise), and did much to help me feel at home in a foreign land by including me in some of his and Elizabeth's special family activities. No matter how discouraged I might have been by a particular aspect of my work, I could always count on Paul to find the hidden morsels of progress worth celebrating and pursuing. From this experience, I have a template for the kind of supervisor I hope I will be when I have PhD students. In addition to our rich personal collaboration, Paul also made it possible for me to take visiting positions at SRI International, Microsoft Research and INRIA. Without his support in this, much of my thesis work would not have come to fruition. I am very pleased we have received a four-year EPSRC grant to continue this work so that our collaboration shall go on for many years to come.

I thank Leonardo de Moura of Microsoft Research, Washington. Leo's friendship has had an immeasurable impact on me and on the work contained in my thesis. It was Leo's idea to focus on developing Gröbner basis methods tailored to the needs of industrial-strength SMT solving, and this goal of his (which has since become also a goal of mine) led us down beautiful paths, many of which are still unwinding. The countless days and nights we spent developing these techniques, making and revising conjectures, and at last proving our long sought after theorems remain with me as some of my favourite memories of my life. I find the sheer intensity of our collaboration impossible to describe. As we are in the throws of writing two more papers together as I type this, I am happy that my long collaboration with Leo is only just beginning.

I thank N Shankar and Sam Owre of SRI International. They are incredible teachers who have given me so much. The first idea for my **RAHD** system for making existential decisions over real closed fields was developed while I was a Visiting Fellow at SRI under Shankar and Sam during May - October, 2008. During this visit, I wrote the initial **RAHD** prototype as an extension of the proof assistant PVS. Shankar taught me much about research and how to place it in the context of an artful and fulfilling life. Sam taught me a tremendous amount about almost everything — his lessons on Lisp, jazz, racquetball, go and snooker remain especially vibrant in my mind. There

are many afternoons I wish I could walk into Sam's office to seek his Lisp advice over a gourd of yerba maté while under the spell of a blasting Cecil Taylor record. Paradise.

At SRI, I was inspired by numerous helpful discussions with John Rushby, Bruno Dutertre and Ashish Tiwari. I thank them for this. I also thank the US National Science Foundation and NASA for funding my SRI fellowship. I send a special thank you to the Berkeley SMASH summer mathematics camp for giving me the chance to share with those unfathomably bright high school students the beauty of mathematics (and of real algebraic decision problems, in particular!). At SRI, my friendships with Max Meier and Florent Kirchner were especially edifying. Further, I am grateful to Florent and INRIA/IRISA for funding my month-long position as a Visiting Researcher at INRIA in Rennes, Bretagne, France in April, 2010, where Florent and I began our work of connecting **RAHD** and the proof assistant Coq. Also on the topic of the French, I have benefited very much from conversations with Yves Bertot and Assia Mahboubi and I thank them for their advice and encouragement.

I thank my fellow LFCS PhD students, Julian Guiterrez, Willem Heijltjes, Ohad Kammar, Gavin Keighren and Matteo Mio. Our regular lunches, pints, musical hangs and invigorating conversations have done much to keep me going. I am thankful to Kousha Etessami and Leonid Libkin for their tremendous help, especially in their role on my yearly progress review committee. I also thank Jeff Egger, Alex Simpson, John Longley, Lorenzo Clemente, Ben Kavanagh, Sarah Luger, Annette Leonhard, Gaya Nadarajan and Teresa Llano for their friendship and advice. I am especially thankful to Gianmaria Silvello, whose nine-month visit to Edinburgh — lived equally between our shared office, The Jazz Bar, Dario's and the Film House Cinema — had a tremendous positive impact on me. To Gianmaria, I say one word: *Legendary*.

I thank The Edinburgh Mathematical Reasoning Group for allowing me to undertake my thesis work in such a welcoming and dedicated community. I've benefited from numerous discussions with Alan Bundy, Lucas Dixon, Jacques Fleuriot, Andrew Ireland, Alan Smaill, Ewen MacLean and Phil Scott over the years and I thank them for this. When my PhD student funding ran out and our grant application was still under review, Alan Bundy found a way to fund me on the DReaM group's Platform Grant — without his help, I would have been in serious trouble. I am also grateful to the Scottish Theorem Provers Seminar and look forward to my continued involvement.

Before I began in Edinburgh, I spent the year 2006-2007 at the Mathematical Research Institute in The Netherlands under Jaap van Oosten and Ieke Moerdijk. This year-long Master Class in Mathematical Logic was vital for my mathematical devel-

opment and I thank my teachers at MRI, Jaap van Oosten, Ieke Moerdidk, Henk Barendregt, Herman Guevers, Bas Spitters, Bas Terwijn, Wim Veldman and Albert Visser. I am especially grateful to my friends and fellow students at MRI, David Carchedi, Yves Fomatati, Danko Iliik, Johanny Suarez, Takako Nemoto and Andrew Polonsky. While in Holland, I taught a mechanical theorem proving course with Joost J. Joosten at the Institute for Logic, Language and Computation of University of Amsterdam, and I thank Joost and our remarkable students. My close friendship and collaboration with Joost has done much to fuel me throughout my PhD. To Joost, I say: *TMNFS2K7AB*.

As an undergraduate at the University of Texas at Austin, I have Bob Boyer, Josh Dever, Matt Kaufmann, Greg Lavender, Vladimir Lifschitz, J Moore and Altha Rodin most to thank for guiding and encouraging my mathematical interests. They have made a tremendous impact on my life. I am especially grateful for my deep friendship with Denis Ignatovich; he continues to be a pivotal source of wisdom. I thank Jeremy Avigad for having me in his 2005 NSF Summer School in Proof Theory at the University of Notre Dame. This course was the first time I learned of the decidability of the theory of real closed fields; I was mesmerised by it then and I am mesmerised by it now. I also thank Tom Ball, Daniel Brown, Yuri Gurevich, John Harrison, Rustan Leino and András Salamon for their friendship and many intellectual gifts.

While I was a PhD student, I wrote two albums with my close friend Barry DeBakey which we recorded in Austin, Texas with the incredible help of Dave and Eddy Hobizal. The first album, "Olney Clark," was produced by Eddy and was especially important for me as a musical outlet to document the process of doing my PhD. Our friendships and musical collaborations continue to provide me with crucial inspiration.

The final bits of my thesis were completed at Cambridge University after I began my RA position on our joint Cambridge-Edinburgh EPSRC grant "Automatic Proof Procedures for Polynomials and Special Functions." I am very grateful to Larry Paulson for his advice and encouragement. I thank the Cambridge Automated Reasoning Group as well as Thomas Forster and the Cambridge Set Theory Group for providing such an open and invigorating environment. At Cambridge, I also thank Mike Gordon and my office-mates, James Bridge, Will Denman and Magnus Myreen. Furthermore, I thank my PhD examiners, Daniel Kroening at Oxford and Alan Smaill at Edinburgh for their careful reading, lively discussion and most helpful suggestions.

Finally, I thank my family. To my parents Donna and John, my sisters Starr, Jacque and Skye: You have loved me into being, you have made me who I am. To you and to Erika: I thank you for your love, your kindness and your unwavering belief in me.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

I have benefited greatly from collaboration with my co-authors Paul B. Jackson and Leonardo de Moura. Much work contained herein is a product of such collaborations. At the end of the introductory section of each chapter containing joint work, I provide an explanation of the contributions each co-author made to the work presented in that chapter. When appropriate, I also give references to our published papers in which such work appears.

(*Grant Olney Passmore*)

To Donna and John, Starr, Jacque and Skye.

Since a decision method, by its very nature, requires no intelligence for its application, it is clear that, whenever one can give a decision method for a class K of sentences, one can also devise a machine to decide whether an arbitrary sentence belongs to K.

It often happens in mathematical research, both pure and applied, that problems arise as to the truth of complicated sentences of elementary algebra or geometry. The decision method presented in this work gives the mathematician the assurance that he will be able to solve every such problem by working at it long enough.

– Alfred Tarski, "A Decision Method for Elementary Algebra and Geometry," 1948.

Contents

Chapter 1

Introduction

1.1 Kissing Spheres and a Theorem of Tarski

Our work begins with an astounding theorem of Alfred Tarski.

Theorem 1.1.1. *The elementary theory of real closed fields admits effective elimination of quantifiers [Tar48].*

From this result, the decidability of elementary real and complex algebra and geometry readily follow, and a most tantalising situation arises: In principle, every elementary arithmetical conjecture over finite-dimensional real and complex spaces may be decided simply by formalising the conjecture and asking a computer of its truth. All one needs is the fortitude to implement a decision method, the dedication to express conjectures formally, access to high-powered computing machinery, and the game is won. The world is filled with few marvels this profound. So why then do we still not know how many unit hyperspheres may kiss[1] in five dimensions? Is it 41? 42?

§

The issue is one of complexity. Though decidable, the theory of real closed fields (**RCF**) is fundamentally infeasible. This observation is made precise by a landmark algorithmic complexity result of the 1980s.

[1]The n-dimensional kissing problem asks: Given an n-dimensional unit hypersphere U centered at the origin in $\mathbb{R}^n$, how many other identical hyperspheres may be arranged so that they each "kiss" U (touch U at a single point) without further overlaps? (For a beautiful telling of the problem, see [PZ04].) In principle, the kissing problem may be solved for each dimension n through iterated application of a quantifier elimination algorithm for elementary algebra and geometry, i.e., by the engine Tarski's theorem guarantees us. But, in practice, this approach is hopeless for reasons we soon discuss.

Theorem 1.1.2 (Davenport-Heinz)**.** *There are families of n-dimensional **RCF** formulas of length $O(n)$ whose only quantifier-free equivalences must contain polynomials of degree $2^{2^{\Omega(n)}}$ and of length $2^{2^{\Omega(n)}}$.*

Thus, arithmetical problems (especially those high-dimensional, i.e., many-variable) will not in general be realistically solvable by full **RCF** decision methods. Yet, there are many examples of difficult high-dimensional **RCF** problems solved in mathematical and engineering practice. What is the disconnect?

1. **RCF** problems solved in practice – especially those solved by hand – are most often solved using an ad hoc combination of methods, *not* by a general decision method.

2. **RCF** problems arising in scientific practice commonly have special structural properties dictated by the application domain from which they originated. Such structural properties can often be exploited making such problems more amenable to analysis and pushing them within the reaches of restricted, more efficient variants of known decision methods.

Our dissertation uses these two observations as the basis of a principled *combined* approach to making real algebraic decisions. Key to this work is being practically-minded: No single complete method can scale to a high-dimensional setting. Thus, we propose a methodology for deciding high-dimensional sentences in the $\exists$ fragment of **RCF** based upon combining *sound but possibly incomplete* proof procedures which are effective for (and, in fact, can be *tailored to*) classes of problems arising in practical verification applications. We are especially interested in manners in which *fast, sound but incomplete* procedures can be used to enhance the practical efficacy of *sound and complete* methods such as cylindrical algebraic decomposition (CAD) by, for instance, recognising when certain expensive computations can be avoided.

1.2 In More Depth

In attempting to make real algebraic decision methods scale to high-dimensional settings, we are faced with what seems to be a rather insurmountable obstacle: the full first-order theory of **RCF** has infeasible complexity, and there are currently no known *complete* methods for the $\exists$ fragment of **RCF** which seem to fare better in practice than full **RCF** quantifier elimination. If we restrict our focus to the $\exists$ fragment of

RCF, then there is a key algorithmic complexity result which brings a sliver of optimism towards the dream of practical, scalable decision methods for high-dimensional $\exists$ **RCF** sentences.

Theorem 1.2.1 (Grigor'ev-Vorobjov)**.** *The $\exists$ fragment of **RCF** can be solved in time singly exponential in dimension.*

However, even the apparent good news found in this result — that the $\exists$ fragment of **RCF** has an exponential speed-up over the full first-order theory — is misleading in a practical sense. Analysis by Hong [Hon91] suggests that known singly-exponential algorithms for $\exists$ **RCF** will perform much worse[2] than even *full first-order* quantifier elimination algorithms such as cylindrical algebraic decomposition for all but astronomically large input formulas. As if this were not enough, the complexities of known **RCF** and $\exists$ **RCF** decision methods are dependent primarily upon the *dimension* of their input formulas. In this regard, scaling **RCF** decision methods to high-dimensional settings seems utterly hopeless.

Yet, there is no denying the fact that applying a full quantifier elimination algorithm to decide the unsatisfiability (i.e., falsity over $\mathbb{R}$) of a formula such as

$$\exists x_1, \ldots, x_{100}(x_1 * x_1 + \ldots + x_{100} * x_{100} < 0)$$

is an obvious misappropriation of computational (and temporal) resources. While an example such as this may seem contrived, consider the fact that when an **RCF** decision method is used in the context of formal verification efforts, it is often fed huge collections of machine-generated formulas which may very well be (un)satisfiable for extremely simple reasons.

In addition, problems arising from a particular application domain often share similar structure which traditional general methods will fail to exploit. We have observed these phenomena first-hand with many of the applications users have made of our **RAHD** tool.

[2]There is a very recent development showing promise in the other direction: Galen Huntington's beautiful 2008 Berkeley PhD thesis, "Towards an efficient decision procedure for the existential theory of the reals," has shown that Canny's singly exponential decision method for $\exists$ **RCF**, a procedure not considered by Hong in his analysis (Hong's analysis was in 1991, and Canny's method was first fully published in 1993 [Can93]), can in fact be implemented and made to solve a number of very small (bivariate, quadratic) examples. While a practical implementation of Canny's method is still a long way off, this work leaves one with a compelling optimism towards the possibility that, in contrast to Hong's conclusions in 1991, practically useful singly exponential decision procedures for $\exists$ **RCF** may eventually be realised.

Thus, it seems advantageous to investigate algorithmic proof methods which attempt to make "easy decisions" quickly. And when such easy decisions fail, it will be desirable if the computations undertaken in attempting them could contribute to lessening the workload required of more heavy-weight analysis procedures which may be subsequently applied.

Finally, if one knows in advance that a large collection of "similar" problems will be encountered, it would be desirable to provide mechanisms for *specialising* the approach of the proof procedure to exploit structural aspects of the formula class whenever possible. These concerns give rise to a particular *combined* approach to developing practical proof procedures for $\exists$ **RCF**.

1.2.1 Our Approach

At the highest level, we would like proof procedures for $\exists$ **RCF** which

- scale to problems of realistic size (especially in many variables),
- are customisable for classes of problems with similar structure.

In working to accomplish this, we are faced with a rather wonderful difficulty: there are *many* different approaches to making **RCF** decisions, each with their own strengths and weaknesses. These include

- quantifier elimination by Muchnik sign matrices [Sch04, MO02],
- quantifier elimination by Cohen-Hörmander sign matrices [MH05],
- quantifier elimination by partial cylindrical algebraic decomposition [Bro04],
- quantifier elimination by virtual term substitution [Wei97],
- Positivstellensatz witness search by the Tiwari method [Tiw05a],
- Positivstellensatz witness search by semidefinite programming [Har07],
- interval constraint propagation and related methods [GB06, FHR$^+$07, Neu90, Rat06],
- connected component sampling by Basu-Pollock-Roy **PSPACE** methods [BPR06],
- techniques based on complex triangulation for zero-dimensional systems [CMXY09],

- Nelson-Oppen-like "distributivity-free" combinations of separate decision procedures for the additive and multiplicative fragments [AF06],

- and many others.

We wish to take advantage of this vast variety of powerful (semi-)decision methods. Our general programme then has been to do roughly as follows:

1. Study deeply, implement, and experiment with a number of different approaches to making $\exists$ **RCF** decisions.

2. Develop new variants of these decision methods by devising methods to effectively *combine* them in compelling ways. Such combinations are compelling, for instance, if with them it possible to decide sentences outside of the practical reach of the individual decision methods when they are used in isolation.

3. Build a tool which incorporates the most compelling decision methods we have investigated and provides a framework for developing, investigating and applying new combinational methods.

This programme has resulted both in a number of novel combined decision methods and in a principled approach (based upon a *proof strategy* language) for facilitating the arbitrary combination of a heterogeneous collection of **RCF** decision techniques in a working tool.

1.3 Parallel Strands: SMT and RAHD

Given our pragmatic methodology, creating tools which allow us to experiment with and guide the development of our combined decision methods is crucial. We have done this via two parallel projects, each giving rise to contributions of both a theoretical and applied character, and each roughly comprising half of our thesis.

The first half considers the integration of nonlinear arithmetical techniques — chiefly, Gröbner bases and related polynomial ideal calculations — into a breed of automatic theorem provers called Satisfiability Modulo Theories (SMT) solvers. SMT solvers will orchestrate the combination of this nonlinear arithmetical reasoning with many other proof search methods for theories very different from nonlinear (real or

complex) arithmetic. To do this effectively, SMT solvers require their proof procedures to behave in very special ways. Great care must be taken to tailor Gröbner basis construction algorithms and the like to the peculiar requirements of efficient SMT.

The second half considers the development of a stand-alone tool **RAHD** (Real Algebra in High Dimensions) strictly for $\exists$ **RCF** reasoning. To realise this tool, we propose and implement a hierarchy of novel combined decision methods, and develop techniques enabling verification practitioners to further create their own decision method combinations suitable for their needs. This involves decomposing and parameterising known decision methods such as CAD or the Tiwari Positivstellensatz method in such a way that other user-specified proof procedures are able to contribute to their processing. This theme culminates in a theoretical framework we call *Abstract Partial CAD*, which prescribes a way for user-specified proof procedures to augment CAD-based decision methods.

Let us further discuss these two strands. As we do so, we will keep an eye towards why each of them, while informing and reinforcing the other, requires quite distinct foci and contributions.

1.3.1 Part I: SMT and Gröbner Bases

SMT solvers are sophisticated automatic theorem provers which orchestrate a combination of DPLL-based SAT solving and the application of so-called *theory solvers* (T-solvers) for decidable (usually quantifier-free) elementary theories including linear real arithmetic, bit-vector arithmetic and uninterpreted functions with equality. SMT solvers (e.g., Z3 [MB08], Yices [DdM06] and CVC3 [BT07]) have seen serious academic and industrial uptake, and form the automated reasoning engines for many widely-used program verification tools. To scale to real-world verification efforts, SMT solvers must support the expression of rich verification conditions and be highly efficient in their processing.

Classically, the lack of T-solvers for nonlinear *real* arithmetic has been a pressing problem barring the extension of SMT methods into the verification of programs with nonlinear arithmetical components. However, the special requirements an effective SMT solver places on its T-solvers preclude full $\exists$ **RCF** decision methods from being integrated as T-solvers directly. Fundamentally, complete decision methods are far too computationally expensive for real-world scalable SMT decision loops.

Thus, to effectively integrate $\exists$ **RCF** reasoning into an SMT solver, it is sensible

to look for *fast, sound but incomplete* proof procedures which are nevertheless effective on classes of problems arising in practice. Luckily, a number of such techniques have been recently put forth: Tiwari and Harrison have proposed methods based upon the Positivstellensatz [Tiw05a, Har07], Jackson and myself have investigated combinations of full-dimensional CAD and Gröbner bases [PJ09] (cf. **Chapter 8**), and Platzer, Quesel and Rümmer have given methods based on Real Nullstellensatz search [PQR09].

Interestingly, all of these new proof procedures require or can be seriously enhanced by Gröbner basis methods taken over the *complex* numbers. Thus, for the goal of obtaining robust nonlinear *real* arithmetic reasoning in the context of SMT, it seems prudent to first focus upon obtaining efficient methods for nonlinear *complex* arithmetic, in particular the adaptation of Gröbner basis methods to the needs of SMT solvers. This is the nature of our contributions to nonlinear *arithmetics* and SMT: We give a theory of Gröbner basis construction algorithms and use it to prove the correctness of a new class of Gröbner basis algorithms and related methods designed to meet key requirements of efficient T-solvers. What are these requirements?

For the orchestration mechanisms used to combine decision procedures to be effective, SMT solvers require their T-solvers to behave in a manner allowing other aspects of SMT proof search to gain the most benefit from a T-solver's conclusions. For example, when a T-solver proves a conjunctive formula to be unsatisfiable, the T-solver should communicate a minimal subset of the assumptions required to obtain the unsatisfiability. This subset, known as an *unsatisfiable core*, allows the central DPLL engine of the SMT solver to prune more branches of its search tree. In addition, a T-solver should be prepared for formulas with massive numbers of constraints (sometimes tens of thousands), and should work in an incremental manner, e.g., avoiding duplicate proof search effort when it is given a sequence of formulas each sharing some atomic constraints.

We will outline our contributions to these goals shortly. Let us for now look to the second half of our dissertation.

1.3.2 Part II: RAHD and Strategic Proof Procedure Combinations

In the second half of our dissertation, we turn our focus to the development of "standalone" proof procedures for $\exists$ **RCF**. By this we mean proof methods which are not intrinsically tied to the needs of SMT solvers. Instead, we take a much broader view:

Not only do we wish to devise novel, powerful combinations of decision methods and make them available in a working tool, but we also want to provide a platform in which users can synthesise their own combined proof procedures tailored to the types of problems they encounter in practice. Our answer to this goal is realised within our tool **RAHD**.

RAHD is a proof tool for orchestrating and applying a heterogeneous collection of **RCF** proof procedures to decide the satisfiability of nonlinear arithmetical formulas over the real numbers. **RAHD** contains original implementations of a vast array of real algebraic algorithms and decision procedures, and is designed to facilitate the combination of such techniques into custom heuristic proof procedures. This specification of custom proof procedures is done using a simple *proof strategy* language.

In addition to its general use as a platform for building and deploying custom $\exists$ **RCF** proof procedures, **RAHD** can also be seen as a first realisation of our framework of *Abstract Partial CAD*. This framework is a generalisation of the well-known *partial CAD* decision method of Collins and Hong [CH91]. The key idea behind *Abstract Partial CAD* is that once $\exists$ **RCF** proof procedures can be synthesised and tailored as needed to specific problem domains, then these $\exists$ **RCF** proof procedures can be treated as first-class objects and given as parameters to augment the processing of other cooperating decision methods. In particular, CAD-based decision methods can be augmented to allow for this kind of proof procedure parameterisation to influence the *lifting* or *stack construction* phase of CAD. This then allows one to externally exert strategic control over CAD computations. We will be most interested in *fast, sound but incomplete* $\exists$ **RCF** proof procedures which can be given as parameters to a CAD-based decision method and used to recognise when certain expensive CAD computations can be avoided. Crucially, Abstract Partial CAD will apply these fast, sound but incomplete proof procedure parameters in the context of a complete CAD-based decision method in a way that never sacrifices the completeness of the underlying CAD-based method.

RAHD can be used in both interactive and automatic modes. Its interactive mode is designed both to facilitate a practitioner's analysis of $\exists$ **RCF** formulas and to provide a platform in which customised $\exists$ **RCF** proof procedures may be built and applied. The methods provided for interactively exploring the proof search and real solution space allow a user to gain much intuition about the formulas in which they are interested, intuition which can then be used to devise appropriate proof strategies. From this perspective, **RAHD** more closely resembles general-purpose tactic-based proof assistants à la PVS, HOL-Light or Coq than it does normal automatic decision procedures such as

SMT solvers. But, once an appropriate proof strategy is installed, the system can then be used in the same way one would any other push-button decision method. In fact, if a user is satisfied with a proof strategy, the system can be instructed to automatically recompile itself and build a binary executable which invokes only the strategy desired. This allows the system, once tailored to a problem domain, to be cleanly integrated into formal verification tool-chains.

At its core, **RAHD** provides original implementations of many **RCF** decision methods and techniques from algorithmic (real) algebraic geometry. We have chosen to write these ourselves (in the programming language Common Lisp) for a number of reasons. Perhaps it is worth a brief discussion to justify this decision.

First of all, to enable the non-trivial combination of different decision methods, we often found it necessary to decompose them into smaller pieces, exposing many aspects of their internal processing to outside influence. For instance, there would be no way to apply our framework of Abstract Partial CAD — allowing proof procedure parameters to be used internally by a CAD-based decision algorithm to augment CAD construction — and experiment with variants of CAD derived from it if we merely utilised another CAD implementation as a monolithic black-box. One must get inside the core algorithms to build in such hooks, and writing these generalised algorithms ourselves seemed much easier than trying to decompose an already existing partial CAD implementation. Similarly, we want to make use of the Gröbner basis insights found during our SMT investigations within **RAHD**. This of course requires their implementation.

Thus, it seemed quite clear that to really build the platform we desired, enabling deep combinations between different proof procedures, we needed to implement the core algorithms ourselves. This has a nice pedagogical byproduct: Namely, we often found it difficult to gain a deep understanding of known proof procedures, and without fail, our understanding was always greatly enhanced by actually implementing them. When it comes to truly learning deep mathematical ideas, we are, after finishing this dissertation, of the resolute opinion that there is no substitute for undertaking implementations of concrete algorithmic aspects of the ideas and experimenting with them heavily.

Finally, given our goals, one may wonder why we did not go a different route: Why was **RAHD** not developed within a general-purpose proof assistant? Ideally, it would have been. But, the mathematics underlying some of the techniques we wished to exploit (e.g., CAD) are deep and difficult to formalise. For instance, Mahboubi's

beautiful 2006 PhD dissertation involved programming CAD within Coq and verifying some background algebra and elimination theory including the theory of subresultants [Mah06]. Completing this work so that the full CAD procedure is proved correct involves serious formalisation, especially for the justification of a CAD projection operator. This work could certainly be the topic of a number of future remarkable PhD theses. But, we wished to go another route.

Our goal is to radically enhance the scope of $\exists$ **RCF** problems solvable in practice. That is, we want to develop new approaches which make practical impact, and to do this, we require much freedom to experiment. We could not commit to verifying the correctness of a new, practically useful decision method before we knew what it was. Thus, we decided to postpone the goal of formally verifying our proof techniques within a general-purpose proof assistant until after we had converged upon methods which truly were compelling. To find these methods, we needed to develop a tool. Hence, **RAHD** evolved in the way it did. We should note that preliminary work has begun on making restricted classes of **RAHD** *proofs* replayable within foundational proof assistants [KP10]. This happens by **RAHD** generating proof traces which are interpreted and elaborated by the proof assistant, rather than through the formal verification[3] of **RAHD** itself. But, this work is in its infancy. The verification of **RAHD** and its proof traces are long-term goals beyond the scope of this thesis.

1.4 Contributions

Our main contributions are as follows.

- Contributions to decision procedures for nonlinear arithmetic over $\mathbb{C}$

    1. A theory of Gröbner basis procedures (Abstract Gröbner Bases) based upon the Bachmair-Dershowitz theory of Abstract Completion and designed to aid the analysis of superfluous S-polynomial criteria w.r.t. arbitrary correct basis construction strategies.
    2. A generalisation of the notion of S-polynomial superfluousness in terms of proof normalisation.

[3] There is an exception: We will see in **Chapter 6** that one key aspect of the **RAHD** source code, the bulk of our machinery for generalised interval constraint propagation, has been formally verified.

3. Proofs of the strategy-independent admissibility of three superfluous S-polynomial criteria using the above generalisation.
4. Gröbner basis construction algorithms based on the OTTER and DISCOUNT saturation loops prominent in first-order automated theorem proving. These algorithms were designed to address the needs of SMT solvers to compute Gröbner bases arising from *large, largely linear* nonlinear polynomial constraint systems in the context of industrial software verification. These algorithms are proved correct using Abstract Gröbner Bases and are thus able to make use of the superfluous S-polynomial criteria referenced above. They have been implemented by our co-author de Moura in the SMT solver Z3 [MB08] and preliminary experimental results obtained by our co-author Jackson are presented. These techniques form the basis of the current nonlinear reasoning mechanisms of Z3.
5. Algebraic machinery and algorithms for eliminating redundancy in weak Nullstellensatz proofs of complex unsatisfiability in the context of SMT solvers. This contributes to the minimisation of such proofs to *unsatisfiable cores*.

- Contributions to decision procedures for nonlinear arithmetic over $\mathbb{R}$

1. The development of a large arsenal of *combinable* heterogeneous $\exists$ **RCF** proof procedures. These include generalised interval constraint propagation, extensions of the Tiwari Positivstellensatz method, and a large number of other saturation and simplification techniques based on parametric discriminants, root bounds, Dolzmann degree shifts, and more. This results in a hierarchy of increasingly powerful combined (and *easily combinable*) proof procedures.
2. The theoretical framework of Abstract Partial CAD, a generalisation of partial CAD which allows arbitrary sound but possibly incomplete $\exists$ **RCF** proof procedures to be given as first-class functional parameters for "short-circuiting" expensive computations during the lifting phase of CAD. This gives one the ability to exert *strategic control* over key aspects of CAD construction.
3. The implementation of the above techniques, together with machinery for

facilitating their combination, in our proof tool **RAHD** (Real Algebra in High Dimensions). **RAHD**'s *proof strategy language* is used to allow users to synthesise their own combined decision methods tailored to their needs.

4. An identification the functional proof procedure parameters of Abstract Partial CAD with **RAHD** proof strategies. This allows us to implement the Abstract Partial CAD framework in **RAHD**, which we do for the case of full-dimensional cell decompositions (using the Brown-McCallum projection operator). Throughout, we pay close attention to how **RAHD** may be trustworthily extended and tailored to exploit structural properties arising in specific problem domains.

5. An empirical investigation into combining full-dimensional CAD and Gröbner basis calculations to extend the use of full-dimensional CAD from $\wedge, \vee$ combinations of *strict* inequalities to those involving equations and inequalities which are non-strict.

6. An empirical investigation into a concrete instance of our Abstract Partial CAD framework which uses interval-based techniques to reduce the number of cells one must lift over during partial CAD construction. This investigation is done w.r.t. full-dimensional cell decompositions and the Brown-McCallum projection operator.

1.5 How to Read This Dissertation

This dissertation is composed of nine chapters and two appendices. There is also a large software system, **RAHD**, which can be obtained (cf. **Appendix A**). Six of these chapters (**Chapters 3 - 8**) consist of original contributions. The other three are this introduction, a chapter on mathematical preliminaries (**Chapter 2**) and the final conclusion (**Chapter 9**).

Discussion of related work is interleaved throughout the chapters. In each chapter containing original research, we begin with an overview of the nature of our contributions. When results were obtained with collaborators, we explain roughly the division of labours and give reference to our relevant publications.

In **Figure 1.5**, we present the high-level dependencies between chapters. We use a box (only for **Chapter 8**) to mean that this chapter can be read independently as a user's manual for our **RAHD** system. We can imagine a practically-minded user

beginning with this chapter and only referring to previous chapters when he finds it necessary to understand the mathematics underlying aspects of the system.

As our thesis consists of two rather independent strands (SMT and Gröbner bases on the one hand, combined **RCF** procedures and **RAHD** on the other), there are a number of approaches a reader could take depending on his interests.

Let us characterise some possible readers and provide them with a roadmap.

- A reader interested in our entire thesis should, of course, read everything from start to finish.
- A reader interested in Gröbner bases and SMT, theory and practice should read **Section 2.1.1**, **Section 2.3**, **Chapter 3**, **Chapter 4** and **Chapter 5**.
- A reader interested in Gröbner bases and SMT, practice should read **Chapter 5** and refer back to **Chapter 3**, **Section 2.1.1**, **Section 2.3** and **Chapter 4** as required.
- A reader interested in combined $\exists$ **RCF** proof procedures, theory and practice should read **Chapter 6, Chapter 7, Chapter 8** and refer back to **Section 2.2, Section 2.1** and **Section 2.3** as required.
- A reader interested in combined $\exists$ **RCF** proof procedures, practice should read **Chapter 8** and refer back to **Chapter 6**, **Chapter 7**, **Section 2.2**, **Section 2.1** and **Section 2.3** as required. This would be appropriate for a user simply wishing to experiment with our **RAHD** system, for instance.
- A reader interested in the framework of Abstract Partial CAD should read **Chapter 7** and then refer to **Chapter 6** and **Chapter 8** as needed.
- A reader interested in a self-contained proof that both the elementary theory of algebraically closed fields of characteristic zero and the theory of real closed fields admit elimination of quantifiers should read **Section 2.1** and **Section 2.2**.
- A reader interested in a brief introduction to Gröbner basis theory from the rewriting perspective should read **Section 2.3** and refer to **Section 2.1** as required.

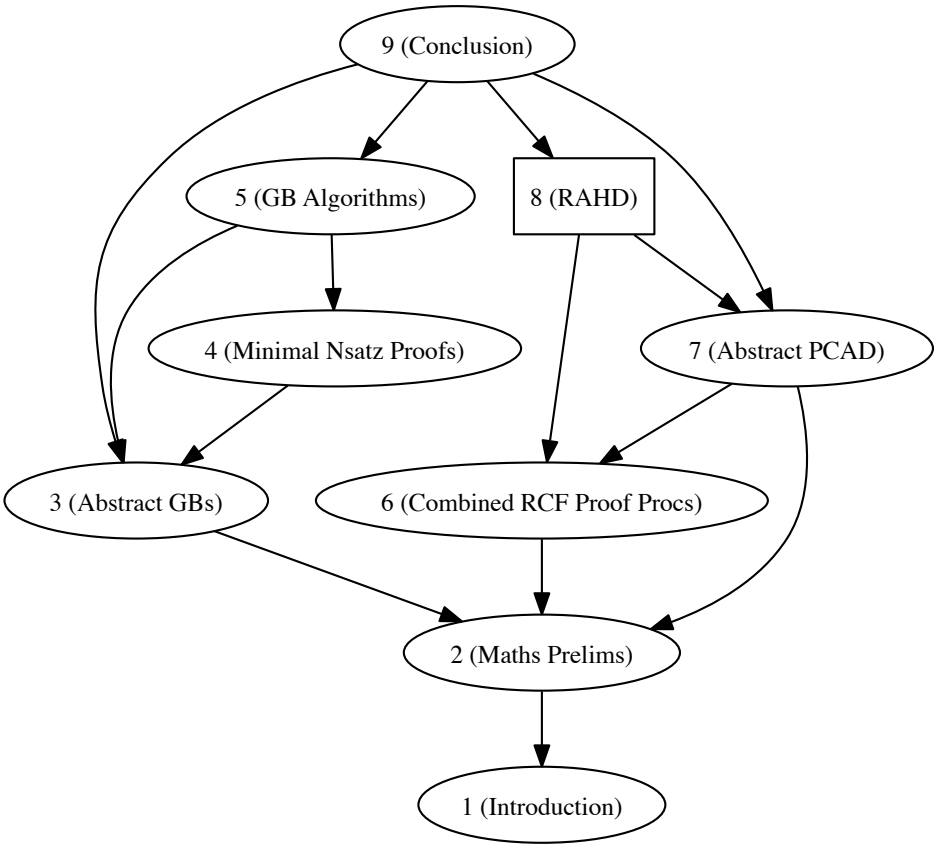

Figure 1.1: Chapter Dependencies

Chapter 8 can be read more-or-less independently as a manual for our **RAHD** system, with the reader when necessary referring to previous chapters for the mathematics underlying the decision techniques present in the system.

Chapter 2

Mathematical Preliminaries

Assumptions

We assume the reader has a grounding in mathematical logic and commutative algebra. We do not however assume exposure to any (real or complex) algebraic geometry and give a self-contained treatment of the relevant (all pre-Grothendieck) foundations.

2.1 Algebraically Closed Fields

2.1.1 Ideals and Affine Varieties

In classical algebraic geometry, the most basic geometric objects of interest are *affine varieties*, which are the sets ("loci") of simultaneous complex zeros of systems of polynomial equations. The descriptive limitation of such objects as being the zeros of polynomial *equations* is no accident — it is forced upon us by the fact that $\mathbb{C}$ admits no ordering as an *ordered field*. The fundamental notion connecting algebra and geometry in this context is the relationship between varieties and ideals.

Let $\mathbb{Q}[\vec{x}] = \mathbb{Q}[x_1, \ldots, x_n]$ denote the ring of polynomials with coefficients in $\mathbb{Q}$ and $n \in \mathbb{N}$ indeterminates $x_1, \ldots, x_n$. Recall the defining property of a polynomial ideal.

Definition 2.1.1 (Polynomial ideal)**.** $\mathcal{I} \subseteq \mathbb{Q}[\vec{x}]$ is a polynomial ideal iff

$$\begin{aligned} 0 &\in \mathcal{I}, \\ p, q \in \mathcal{I} &\Rightarrow p + q \in \mathcal{I}, \\ p \in \mathcal{I},\ q \in \mathbb{Q}[\vec{x}] &\Rightarrow pq \in \mathcal{I}. \end{aligned}$$

We then have the following basic relationship between varieties and ideals.

Definition 2.1.2 (Ideal of a variety)**.** Given a complex affine variety

$$\mathcal{V}_{\mathbb{C}}(S) = \{\vec{c} \in \mathbb{C}^n \mid \forall p \in S\ (p(\vec{c}) = 0)\},$$

with $S \subseteq \mathbb{Q}[\vec{x}]$, $\mathcal{V}_{\mathbb{C}}(S)$ gives rise to a corresponding algebraic object,

$$\mathcal{I}(\mathcal{V}_{\mathbb{C}}(S)) = \{p \in \mathbb{Q}[\vec{x}] \mid \forall \vec{c} \in \mathcal{V}_{\mathbb{C}}(S)\ (p(\vec{c}) = 0)\},$$

the *ideal of polynomials vanishing on* $\mathcal{V}_{\mathbb{C}}(S)$.

Definition 2.1.3 (Ideal generated by polynomials)**.** Given a finite collection of polynomials $S = \{p_1, \ldots, p_k\} \subseteq \mathbb{Q}[\vec{x}]$, we define the *ideal generated by S*, written $\mathcal{I}(S)$, as

$$\mathcal{I}(S) \;=\; \mathcal{I}(\{p_1, \ldots, p_k\}) \;=\; \left\{ \sum_{i=1}^{k} p_i q_i \mid q_i \in \mathbb{Q}[\vec{x}] \right\}.$$

We then have by construction,

$$\mathcal{I}(\mathcal{V}_{\mathbb{C}}(S)) \supseteq \mathcal{I}(S).$$

Observation 2.1.4. It is important to note that these two objects above are not in general equal. Consider $S = \{x^2, y^2\}$, noting that $x, y \notin \mathcal{I}(S)$. Then, $\mathcal{V}_{\mathbb{C}}(S) = \{(0,0)\}$ and so $\{x, y\} \subset \mathcal{I}(\mathcal{V}_{\mathbb{C}}(S))$, yielding $\mathcal{I}(S) \subset \mathcal{I}(\mathcal{V}_{\mathbb{C}}(S))$.

We now turn to an important property of ideals in polynomial rings over Noetherian rings of coefficients. For concreteness, we state the result only for $\mathbb{Q}[\vec{x}]$.

Theorem 2.1.5 (Hilbert's Basis Theorem)**.** *Let $\mathcal{I} \subseteq \mathbb{Q}[\vec{x}]$ be a polynomial ideal. Then,*

$$\exists\, b_1(\vec{x}), \ldots, b_k(\vec{x}) \in \mathbb{Q}[\vec{x}] \;\; \textit{s.t.} \;\; \mathcal{I} = \mathcal{I}(\{b_1, \ldots, b_k\}) \;\; (k \in \mathbb{N}).$$

That is, Hilbert's Basis Theorem guarantees an ideal $\mathcal{I} \subseteq \mathbb{Q}[\vec{x}]$ is always finitely generated, provided that $\mathbb{Q}[\vec{x}]$ is a polynomial ring in finitely many indeterminates, which will hold for all polynomial rings in this dissertation. So, over $\mathbb{C}$ we may reduce the definition of $\mathcal{I}(\mathcal{V}_{\mathbb{C}}(S))$ above to one in which each member of the ideal is a sum of products of members of $\mathbb{Q}[\vec{x}]$ and only k-many generating *basis* polynomials. Thus, taking $\{b_1, \ldots, b_k\}$ to be such a basis,

$$\mathcal{I}(\mathcal{V}_{\mathbb{C}}(S)) = \mathcal{I}(\{b_1, \ldots, b_k\}) = \left\{ \sum_{i=1}^{k} b_i q_i \mid q_i \in \mathbb{Q}[\vec{x}] \right\}.$$

With this in mind, we see it is no geometric restriction to require all ideals to be finitely presented. We may now pose a question over $\mathbb{C}$ central to our decision methods of interest:

Question 2.1.6. *Given $S \subset \mathbb{Q}[\vec{x}]$ s.t. $|S| < \omega$, is $\mathcal{V}_{\mathbb{C}}(S)$ empty?*

Note that this question is equivalent to the following perhaps more familiar one:

Question 2.1.7. *Given a finite collection of polynomial equations:*

$$\begin{aligned} p_1(\vec{x}) &= q_1(\vec{x}), \\ &\vdots \\ p_n(\vec{x}) &= q_n(\vec{x}), \end{aligned}$$

does $\langle \mathbb{C}, +, -, *, =, 0, 1 \rangle \models \exists \vec{x}(p_1(\vec{x}) = q_1(\vec{x}) \wedge \ldots \wedge p_n(\vec{x}) = q_n(\vec{x}))$*?*

That is, a decision method for **Question 2.1.6** would result in a decision method for the *satisfiability* of finitely-presented equational polynomial constraints over $\mathbb{C}$. This decision problem does indeed admit an algorithmic solution. In this dissertation, we will consider techniques for solving this problem based both on *quantifier elimination* and on *Gröbner bases*.

In order to prepare the reader for work that follows, we will present an axiomatisation of the elementary theory of algebraically closed fields (**ACF**) and prove the classical result that the theory of algebraically closed fields of *characteristic zero* admits elimination of quantifiers. The detailed presentation of such a proof is important as it introduces concepts which will be helpful in subsequent chapters. Following this, we will then present the relevant foundations of Gröbner basis theory in **Section 2.3**.

2.1.2 Axiomatisation of ACF

We now present an axiomatisation of the elementary theory of algebraically closed fields. We shall then prove that the characteristic zero extension of this theory admits elimination of quantifiers. This result is due, using predominantly syntactic methods, to Tarski [Tar48], though it was subsequently recast in model-theoretic terms by Abraham Robinson in his 1949 PhD thesis [Rob49]. Many approaches to this result have since been developed. We have chosen to present a method due to Muchnik[1] and discuss why we made this choice below.

[1] Unfortunately, it seems Muchnik never published his result. Instead, it was communicated to his students and colleagues and then appeared in two publications in Russian [Sem86, SV00] in which it was attributed to him. We have learned the method from two English reconstructions of Muchnik's approach [Sch04, MO02]. Our exposition is original, however, and develops the method — for better or for worse — in substantially more detail than the sources from which we learned it.

Definition 2.1.8 (Axiomatisation of **ACF**)**.** Let $\mathcal{L}_{\mathcal{R}}$ be the first-order language with constants $\{0,1\}$, relation symbol $\{=\}$, function symbols $\{+,-,*\}$, and logical symbols $\{\wedge,\vee,\neg,\forall,\exists\}$. $\mathcal{L}_{\mathcal{R}}$ is called the *language of rings*. Given an $\mathcal{L}_R$ term t, we use t^n as shorthand for $t*t*\ldots*t$ and $(t \neq 0)$ as shorthand for $\neg(t=0)$. As no ambiguity will arise, we use $=$ as both object-theoretic and meta-theoretic equality. **ACF**, the elementary theory of algebraically closed fields, is then the $\mathcal{L}_{\mathcal{R}}$-theory defined as:

$$\mathbf{ACF} = \mathbf{F} \bigcup \mathbf{U},$$

where

1. **F** is an axiomatisation of the elementary $\mathcal{L}_{\mathcal{R}}$-theory of *fields*,
2. $\mathbf{U} = \{\forall a_0,\ldots,a_{n-1}\exists x(a_0+a_1*x+a_2*x^2+\ldots+a_n*x^n=0) \mid n\in\mathbb{N} \text{ s.t. } n\geq 1\}$.

Note that **U**, the collection of $\mathcal{L}_{\mathcal{R}}$-sentences stating that every univariate non-constant polynomial with coefficients in the field has a root in the field, is a countably infinite first-order axiom scheme. Note also that in the case of **F**, field properties are readily expressed in $\mathcal{L}_{\mathcal{R}}$ by eliminating multiplicative inverses in favour of their defining multiplicative property (e.g., x^{-1} is replaced with a fresh variable y and the constraint $x*y=1$ is conjoined with y quantified as is contextually appropriate). This is done so that every function symbol in $\mathcal{L}_{\mathcal{R}}$ denotes a total function. The observant reader may notice an apparent impoverishing of our term language by taking polynomials in $\mathbb{Z}[\vec{x}]$ as opposed to $\mathbb{Q}[\vec{x}]$ as in the previous pages. This is of course not a real restriction of expressibility as equations between polynomials in $\mathbb{Q}[\vec{x}]$ may be transformed by multiplying through denominators to semantically (i.e., field-theoretically) equivalent equations between polynomials in $\mathbb{Z}[\vec{x}]$.

As it stands, **ACF** is not a complete theory. This is because the *characteristic* of the field is not specified. For instance, in the absence of a specified characteristic, the ground sentence $(1+1=0)$ cannot be decided. If we specify a characteristic to obtain a theory $\mathbf{ACF}_p$ of algebraically closed fields of characteristic p, then $\mathbf{ACF}_p$ is complete, decidable, and admits elimination of quantifiers. As our interest in algebraically closed fields is chiefly motivated by making decisions over the complex numbers, we henceforth deal only with algebraically closed fields of characteristic zero.

2.1.3 $\mathbf{ACF}_0$ Admits Quantifier Elimination

We shall now prove the important theorem on quantifier elimination which will lead to the decidability of $\mathbf{ACF}_0$. Geometrically, it is essentially the theorem of Chevalley

stating projections of constructible sets are themselves constructible, though proved effectively by presenting an algorithm for obtaining explicit descriptions of such projections as constructible sets. When proving this general quantifier elimination result about $\mathbf{ACF}_0$, we will often reason concretely over the complex numbers. Each time this is done, however, the reader should observe that the properties of $\mathbb{C}^n$ actually used in fact hold over *every* algebraically closed field of characteristic zero, and so our reasoning carries over to the theory $\mathbf{ACF}_0$ as a whole.

We prove this theorem by presenting the complex specialisation of a real quantifier elimination procedure due originally to Muchnik. This procedure is very elementary compared to those we will encounter in subsequent chapters, and is of limited practical interest. But, its simple nature makes it pedagogically superior to other more advanced methods, and it provides a vehicle for introducing many important concepts which will be needed for decision procedures covered later. It also has the advantage that much of the algebraic machinery we define in the context of this $\mathbf{ACF}_0$ result will be reusable in the **RCF** case in the next section where we present the Muchnik procedure in its full **RCF** form.

The result we will prove is as follows.

Theorem 2.1.9 ($\mathbf{ACF}_0$ Quantifier Elimination)**.** *The theory of algebraically closed fields of characteristic zero admits effective elimination of quantifiers.*

We prove this by induction by showing how to eliminate a single existential quantifier from a formula with parameters. First, we introduce some algebraic machinery.

2.1.3.1 Complex Root Diagrams

Definition 2.1.10 (Labeled Row of Roots)**.** If $p \in \mathbb{Z}[x]$ and $C \subset \mathbb{C}$ s.t. $|C| < \omega$, then a $(1 \times |C|)$ labeled binary matrix α with its columns uniquely labeled by members of C is a *labeled row of roots* for p iff

- $p \neq 0 \Longrightarrow$
  $[(\exists \zeta \in C \text{ s.t. } \alpha(\zeta) = 1) \wedge (|\{\zeta \in C \mid \alpha(\zeta) = 0\}| = |\{\zeta \in \mathbb{C} \mid p(\zeta) = 0\}|)]$,
- $\forall \zeta \in C\ (\alpha(\zeta) = 0 \iff p(\zeta) = 0)$.

Note that since α is a row matrix with its columns uniquely labeled by members of C, we use $\alpha(c)$ for $c \in C$ to mean the binary value α holds in the unique column labeled c (i.e., we are, when convenient, treating α as a function in $\{0,1\}^C$). We write $\mathcal{R}_{\mathbb{C}}(\alpha, p)$ to mean that α is a labeled row of roots for p.

Observation 2.1.11. If α is a labeled row of roots for $p \neq 0$, then α contains precisely as many 0's as there are distinct roots of p.

Observation 2.1.12. If α is a labeled row of roots for $p = 0$, then α is a row of 0's.

Definition 2.1.13 (Root Diagram)**.** If $P = \{p_1, \ldots, p_k\}$ is a set of polynomials in $\mathbb{Z}[x]$ then a *root diagram* for P is a labeled binary matrix M with columns labeled by members of $C \subset \mathbb{C}$ s.t. $|C| < \omega$ and rows labeled $p_1, \ldots, p_k$ s.t.

- $\forall p_i \in P\ (\mathcal{R}_{\mathbb{C}}(M(p_i), p_i))$,
- $\exists \zeta \in C\ \forall p_i \in P\ (M(p_i, \zeta) = 0 \iff p_i = 0)$,

where $M(p_i)$ is the row labeled by p_i. We call columns ζ witnessing the second property above *anti-solutions*, as they correspond to sample points within regions of $\mathbb{C}$ in which no non-zero $p_i \in P$ vanishes.

Observation 2.1.14. If M is a root diagram for P and M' is obtained from M by some combination of

- permuting the columns of M,
- adding or removing some (but not all) anti-solution columns from M,
- choosing a different label for an anti-solution column of M (while still preserving its status as an anti-solution column),

then M' is still a root diagram for P.

Lemma 2.1.15. *Up to the modifications described in **Observation 2.1.14**, a root diagram M for P is uniquely determined.*

Proof. This is immediate, as every root diagram for P must contain a *minimal core* consisting of columns labeled by every root of each $p \in P$, together with at least one anti-solution column for P. It is clear that none of these columns may be removed from M while maintaining its status as a root diagram for P, though a different label may be used for the anti-solution column. Thus, given two distinct root diagrams M, M' for P, M and M' may only differ by the operations given in **Observation 2.1.14**. □

Given this relative uniqueness, we will now write $\mathcal{D}(P)$ ("**the** root diagram for P") to mean a canonically chosen root diagram for P.

2.1.3.2 Muchnik Sets and Sequences

We now present the concepts of Muchnik sets and sequences which we will use to compute root diagrams for sets of polynomials.

Let $\mathbb{A}$ be a unique factorisation domain (UFD) and let $p \in \mathbb{A}[x]$ s.t.

$$(p = 0) \ \vee \ (p = \sum_{j=0}^{d} c_j x^j \ \wedge \ c_d \neq 0).$$

Definition 2.1.16 (Polynomial Degree)**.**

$$deg(p) = \begin{cases} d & \text{if } p \neq 0, \\ -\infty & \text{if } p = 0. \end{cases}$$

Definition 2.1.17 (Polynomial Tail)**.**

$$\tau(p) = \begin{cases} \sum_{j=0}^{d-1} c_j x^j & \text{if } p \neq 0, \\ 0 & \text{if } p = 0. \end{cases}$$

We now face a problem: we will need to perform division upon pairs of polynomials in $\mathbb{A}[x]$, where $\mathbb{A}$ is some non-Euclidean UFD such as $\mathbb{Z}[y_1, y_2]$. Recall that if $\mathbb{A}$ is a UFD, then $\mathbb{A}[x]$ is as well. We will thus make use of *polynomial pseudo-division*, as the unique *pseudo-remainder* it computes for pairs of polynomials will be sufficient for inductively obtaining root diagrams.

Let $q \in \mathbb{A}[x]$ s.t.

$$q = \sum_{j=0}^{e} b_j x^j \ \wedge \ q \neq 0 \ \wedge \ deg(q) = e \leq d.$$

Definition 2.1.18 (Polynomial Pseudo-remainder)**.** Given p, q as specified above, *polynomial pseudo-division* of p by q will compute unique $h, r \in \mathbb{A}[x]$ s.t.

$$b_e^{d-e+1} p = hq + r \ \wedge \ deg(r) < e.$$

We refer to $r = rem(p, q)$ as the *pseudo-remainder*[2] of p by q.

Let $\mathbb{M} \neq \emptyset \subset \mathbb{A}[x]$ s.t. $|\mathbb{M}| < \omega$.

Definition 2.1.19 (Muchnik Set)**.** We say $\mathbb{M}$ is a *Muchnik set* iff

[2] Knuth gives an excellent presentation of polynomial pseudo-division on pp 425-428 of [Knu97]. Sufficient background on UFDs is given on pp 422-424. To reiterate the elementary nature of the quantifier elimination procedure we are presenting, though, let us note that computing pseudo-remainders is conceptually very simple: In fact, over a UFD such as our $\mathbb{A}[x]$ above, one can compute the pseudo-remainder of p by q by first multiplying p by b_e^{d-e+1} and then performing standard polynomial division (that is, the division algorithm for polynomials over a field) between the product $(b_e^{d-e+1} p)$ and q.

1. $p \in \mathbb{M} \Longrightarrow \tau(p) \in \mathbb{M}$,
2. $p \in \mathbb{M} \Longrightarrow \frac{\partial p}{\partial x} \in \mathbb{M}$,
3. $p, q \neq 0 \in \mathbb{M} \wedge deg(q) \leq deg(p) \Longrightarrow rem(p,q) \in \mathbb{M}$.

Observation 2.1.20. If $\mathbb{M}$ is Muchnik, then $0 \in \mathbb{M}$.

Definition 2.1.21 (Muchnik Closure)**.** Given $\mathbb{M} \subset \mathbb{A}[x]$, let $\mathbb{M}^*$ be the smallest Muchnik set containing $\mathbb{M}$. We say $\mathbb{M}^*$ is the *Muchnik closure* of $\mathbb{M}$.

Observation 2.1.22. If $\mathbb{M} \subset \mathbb{A}$ s.t. $|\mathbb{M}| < \omega$ and $0 \in \mathbb{M}$ then $\mathbb{M}$ is Muchnik.

Definition 2.1.23 (Constant Fragment)**.** A *constant* is a polynomial $p \in \mathbb{A}$. If $\mathbb{M}$ is Muchnik then let $\mathbb{M}_0$ be $\mathbb{M} \cap \mathbb{A}$ – the *constant fragment* of $\mathbb{M}$ – which is also Muchnik.

Lemma 2.1.24 (Finiteness)**.** *Let* $\mathbb{M} \subset \mathbb{A}[x]$ *s.t.* $|\mathbb{M}| < \omega$. *Then* $|\mathbb{M}^*| < \omega$.

Proof. Immediate as each of the three operations placing polynomials into $\mathbb{M}^*$ are *strictly* degree reducing. □

All Muchnik sets we encounter will be finite and we henceforth omit the explicit assumption. We now introduce the notion of a *Muchnik sequence* and prove its important substructural property.

Definition 2.1.25 (Muchnik Sequence)**.** Let $\mathbb{M}$ be Muchnik. Then any sequence $\beta \in \mathbb{M}^{|\mathbb{M}|}$ is a *Muchnik sequence* iff

$$\forall 1 \leq i < |\mathbb{M}| \ (deg(\beta(i)) \leq deg(\beta(i+1)).$$

Observe that as $deg(0) = -\infty$, a Muchnik sequence always begins with 0.

The following lemma will be important for inductively extending Muchnik sequences.

Lemma 2.1.26 (Muchnik Subsequence)**.** *Let* β *be a Muchnik sequence. Then, any non-empty initial segment* β' *of* β *is a Muchnik sequence.*

Proof. By the definition of Muchnik, we must show β' is closed under the operations of *tail*, *partial differentiation* and *pseudo-remainder*. But this is immediate by the fact that these three closure operations are strictly degree reducing, and in a Muchnik sequence β (and hence any initial segment), the polynomials must be ordered so that $(deg(\beta(i)) \leq deg(\beta(i+1))$. □

2.1.3.3 Elimination of a Single Existential Quantifier

With the Muchnik machinery in hand, let us now discuss our strategy for eliminating an existential quantifier. Given a set of polynomials $P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[y_1, \ldots, y_n][x]$ (e.g., $\mathbb{A} = \mathbb{Z}[\vec{y}]$ is our ambient ring of coefficients for polynomials in x), let $P(\vec{c}) = \{p_1(\vec{c}), \ldots, p_k(\vec{c})\} \subset \mathbb{Z}[x]$ for any $\vec{c} \in \mathbb{C}^n$. Recall that P^* is the Muchnik closure of P with $P_0^* = (P^* \cap \mathbb{Z}[\vec{y}])$ its subset of constants w.r.t. x. We will present an algorithm for computing all possible root diagrams for P^*. This will be done in such a way that each root diagram for P^* will be derived (and uniquely determined) from a root diagram for P_0^*. By viewing P^* as a collection of univariate polynomials in x, the set of all root diagrams for P^* arises by considering the specialisations of $\mathbb{A} = \mathbb{Z}[\vec{y}]$ to points $\vec{c} \in \mathbb{C}^n$. Let us introduce the notion of an *extended root diagram* to formalise this process.

Definition 2.1.27 (Extended Root Diagram)**.** Let $P = \{p_1, \ldots, p_k\}$ be a set of polynomials in $\mathbb{Z}[y_1, \ldots, y_n][x]$. Then, an *extended root diagram* for P w.r.t. $\vec{c} \in \mathbb{C}^n$ is a labeled binary matrix $M \in \{0,1\}^{P \times C}$ with $C \subset \mathbb{C}$ and $|C| < \omega$ s.t.

- $\forall p_i \in P(\mathcal{R}_{\mathbb{C}}(M(p_i), p_i(\vec{c})))$,
- $\exists \zeta \in C \; \forall p_i \in P \; (M(p_i, \zeta) = 0 \iff p_i(\vec{c}) = 0)$,

where $M(p_i)$ is identified with a function in $\{0,1\}^C$ in the obvious way.

Intuitively, if M is an extended root diagram for P w.r.t. $\vec{c}$, then M is in principle a normal root diagram for $P(\vec{c}) \subset \mathbb{Z}[x]$, but constructed so that row the labels of M hold a record of the polynomials $p_i \in \mathbb{Z}[\vec{y}][x]$ from which the $p_i(\vec{c}) \in \mathbb{Z}[x]$ were derived. Thus, one may see such an extended root diagram as what happens when one computes a root diagram for a specialisation of P to $P(\vec{c})$ and then *forgets* the specialisation of the row labels.

It is easy to see that the same uniqueness properties that hold for root diagrams (à la **Lemma 2.1.15**) also hold for extended root diagrams.

Finally, it turns out that all of the information needed to perform quantifier elimination can actually be obtained from a variant of extended root diagrams in which neither the columns nor rows carry explicit labels.

Definition 2.1.28 (Unlabeled Extended Root Diagram)**.** Let $P = \{p_1, \ldots, p_k\}$ be a set of polynomials in $\mathbb{Z}[y_1, \ldots, y_n][x]$. Then, an *unlabeled extended root diagram* for P w.r.t. $\vec{c} \in \mathbb{C}^n$ is an $(k \times m)$ binary matrix M obtained from an extended root diagram M' for P w.r.t. $\vec{c}$ by forgetting the row and column labels of M'. That is, M is simply

the underlying binary matrix of M'. Given that P is ordered, we will use $M(p_i)$ and "the row corresponding to p_i" to mean the ith row of M, even though M is formally simply a matrix (without explicit polynomial row labels). If M is a matrix consisting of a single column, then we will use $M(p_i)$ to mean the value of the single entry in the row corresponding to p_i.

From now on, when we say "**the** unlabeled extended root diagram for P w.r.t. $\vec{c}$," we will mean a *canonically chosen* unlabeled extended root diagram for P w.r.t. $\vec{c}$. We will write $\mathcal{D}^*(P,\vec{c})$ to mean the unlabeled extended root diagram for P w.r.t. $\vec{c}$. Similarly, if β is a Muchnik sequence of polynomials in $\mathbb{Z}[\vec{y}][x]$, then $\mathcal{D}^*(\beta,\vec{c})$ will be the unlabeled extended root diagram for the underlying set of β w.r.t. $\vec{c}$.

Let

$$\mathbb{D} = \{\mathcal{D}^*(P^*,\vec{c}) \mid \vec{c} \in \mathbb{C}^n\},$$

and

$$\mathbb{D}_0 = \{\mathcal{D}^*(P_0^*,\vec{c}) \mid \vec{c} \in \mathbb{C}^n\}.$$

The key observations are:

1. Both sets $\mathbb{D}$ and $\mathbb{D}_0$ are *finite* (and every member of $\mathbb{D}_0$ consists of a *single-column* binary matrix), and

2. Given any $\vec{c} \in \mathbb{C}^n$, the unlabeled extended root diagram for P^* w.r.t. $\vec{c}$ (i.e., $\mathcal{D}^*(P^*,\vec{c}) \in \mathbb{D}$) may be obtained from the unlabeled extended root diagram for P_0^* w.r.t. $\vec{c}$ (i.e., $\mathcal{D}^*(P_0^*,\vec{c}) \in \mathbb{D}_0$).

This derivation of $\mathcal{D}^*(P^*,\vec{c})$ from $\mathcal{D}^*(P_0^*,\vec{c})$, which we call *diagram lifting*, will be done by an algorithm $\mathcal{A}_{\mathbb{C}}$ with the following universal property:

$$\forall \vec{c} \in \mathbb{C}^n(\mathcal{A}_{\mathbb{C}}(\mathcal{D}^*(P_0^*,\vec{c})) = \mathcal{D}^*(P^*,\vec{c})).$$

Let us now see how this machinery can be applied.

Consider a quantifier-free formula

$$\varphi(\vec{y},x) = (\bigwedge_{i=1}^{k}(p_i\ \sigma_{p_i}\ 0)) \quad \text{with} \quad (\sigma_{p_i} \in \{=,\neq\}).$$

Let $Z(\sigma_{p_i})$ hold iff σ_{p_i} is '='. Say that the unlabeled extended root diagram $\mathfrak{C}$ for P^* is φ-*compatible* iff there exists a column j in $\mathfrak{C}$ s.t.

$$\forall 1 \leq i \leq k(\mathfrak{C}(p_i,j) = 0 \iff Z(\sigma_{p_i})).$$

Given any $\vec{c} \in \mathbb{C}^n$, we will then have

$$\exists x(\varphi(\vec{c},x)) \iff \mathcal{A}_{\mathbb{C}}(\mathcal{D}^*(P_0^*,\vec{c})) \text{ is } \varphi\text{-compatible.}$$

Let k_0 be s.t. $P_0^* = \{p_1,\ldots,p_{k_0}\}$. Observe that $|\mathbb{D}_0| \leq 2^{k_0}$. That is, there are at most 2^{k_0} possible unlabeled extended root diagrams which could arise in the process of specialising P_0^* to any point in $\mathbb{C}^n$. Let $\mathcal{M}_0$ be the set of all $(k_0 \times 1)$ binary matrices. Observe that $\mathbb{D}_0 \subseteq \mathcal{M}_0$. Then, conditions $\psi(\vec{y})$ upon $\vec{y}$ s.t.

$$\forall \vec{y}(\psi(\vec{y}) \iff \exists x(\varphi(\vec{y},x)))$$

are given by

$$\psi(\vec{y}) = \bigvee_{d_0 \in \mathcal{Q}_\varphi} \mathcal{E}_{\mathcal{L}_R}(d_0),$$

where

$$\mathcal{Q}_\varphi = \{d_0 \in \mathcal{M}_0 \mid \mathcal{A}_{\mathbb{C}}(d_0) \text{ is } \varphi\text{-compatible}\}$$

and

$$\mathcal{E}_{\mathcal{L}_R}(d_0) = \bigwedge_{q \in P_0^*} (q \ \odot_{d_0(q)} \ 0)$$

s.t.

$$\odot_{d_0(q)} = \begin{cases} \text{`}=\text{'} & \text{if } d_0(q) = 0, \\ \text{`}\neq\text{'} & \text{if } d_0(q) = 1. \end{cases}$$

Now, there is one aspect of the above derivation of $\psi(\vec{y})$ which is counterintuitive. Naively, one would expect $\mathcal{Q}_\varphi$ to be defined as the set S as follows:

$$\mathcal{S} = \{d_0 \in \mathbb{D}_0 \mid \mathcal{A}_{\mathbb{C}}(d_0) \text{ is } \varphi\text{-compatible}\}.$$

The issue with this definition is that in practice, we will not a priori know if a given binary matrix $d_0 \in \mathcal{M}_0$ is actually the unlabeled extended root diagram for P_0^* w.r.t. any $\vec{c} \in \mathbb{C}^n$. That is, given some $d_0 \in \mathcal{M}_0$, we will not know in advance if it is a member of $\mathbb{D}_0$ or not. Thankfully, it will will not actually matter what our lifting algorithm gives as the value of $\mathcal{A}_{\mathbb{C}}(d_0)$ when $d_0 \in (\mathcal{M}_0 \setminus \mathbb{D}_0)$. Let us see why this is so.

Lemma 2.1.29. *Let $d_0 \in (\mathcal{M}_0 \setminus \mathbb{D}_0)$. Then, based upon our construction of $\psi(\vec{y})$ above, it does not matter which $(k \times m)$ binary matrix our lifting algorithm constructs as the value $\mathcal{A}_{\mathbb{C}}(d_0)$.*

Proof. Assume we are eliminating $\exists x$ from $\exists x \varphi(\vec{y},x)$ to obtain a quantifier-free equivalent formula $\psi(\vec{y})$ as above. Consider $d_0 \in (\mathcal{M}_0 \setminus \mathbb{D}_0)$. That is, d_0 is a $k_0 \times 1$ binary

matrix that is not realisable as the unlabeled extended root diagram for P_0^* w.r.t. any $\vec{c} \in \mathbb{C}^n$. Now, let us apply $\mathcal{A}_{\mathbb{C}}$ to lift d_0 and obtain an $(k \times m)$ binary matrix $d = \mathcal{A}_{\mathbb{C}}(d_0)$. We have two cases: either d is φ-compatible, or it is not. If d is not φ-compatible, then d_0 will not contribute at all to our construction of $\psi(\vec{y})$ and so the value of d does not matter. On the other hand, assume that d is φ-compatible. Then, d will contribute to our construction of $\psi(\vec{y})$ in the following way: $\mathcal{E}_{\mathcal{L}_R}(d_0)$ will be present as a disjunct in $\psi(\vec{y})$. But, since d_0 is not realisable as the unlabeled extended root diagram of P_0^* w.r.t. any $\vec{c} \in \mathbb{C}^n$, this means that

$$\langle \mathbb{C}, +, -, *, 0, 1 \rangle \models \forall \vec{y} (\mathcal{E}_{\mathcal{L}_R}(d_0) \iff 0 = 1).$$

Thus, $\mathcal{E}_{\mathcal{L}_R}(d_0)$ is only contributing a contradictory conjunction as a disjunct in our formula $\psi(\vec{y})$, which means $\psi(\vec{y})$ is logically equivalent to $\psi(\vec{y})$ with $\mathcal{E}_{\mathcal{L}_R}(d_0)$ removed. So, it is indeed the case that if d_0 is not realisable as the unlabeled extended root diagram of P_0^* w.r.t. any $\vec{c} \in \mathbb{C}^n$, then it does not matter which $(k \times m)$ binary matrix our lifting algorithm constructs as the value $\mathcal{A}_{\mathbb{C}}(d_0)$. □

This fact permits us a simple approach to constructing $\psi(\vec{y})$: We will generate *all* 2^{k_0} possible $(k_0 \times 1)$ binary matrices as *candidate* unlabeled extended root diagrams for P_0^*, lift each of them, and construct $\psi(\vec{y})$ as a disjunction of conjuncts corresponding to the φ-compatible lifted candidates.

Thus, once we have exhibited an algorithm $\mathcal{A}_{\mathbb{C}}$ for *diagram lifting*, we will have proved the following theorem establishing, by induction, that $\mathbf{ACF}_0$ admits elimination of quantifiers.

Theorem 2.1.30 (Projective Closure of Definability)**.** *Given any quantifier-free $\mathcal{L}_R$-formula $\varphi(\vec{y}, x)$ there exists a quantifier-free $\mathcal{L}_R$-formula $\psi(\vec{y})$ s.t.*

$$\boldsymbol{ACF}_0 \models \forall \vec{y} \ (\exists x \varphi(\vec{y}, x) \iff \psi(\vec{y})).$$

Moreover, $\psi(\vec{y})$ is effectively computable from $\varphi(\vec{y}, x)$.

Let us now finish the proof by constructing such an $\mathcal{A}_{\mathbb{C}}$. Recall that by **Lemma 2.1.26**, every subsequence β' of β is Muchnik. Given an unlabeled extended root diagram for β_0 w.r.t. $\vec{c}$, $\mathcal{A}_{\mathbb{C}}$ will use this property of Muchnik sequences to build an extended root diagram for β w.r.t. $\vec{c}$ inductively, by building one for each of its subsequences β'. We now construct an algorithm $\mathcal{A}^1_{\mathbb{C}}$ which will handle the inductive step of this lifting process.

Lemma 2.1.31 (**ACF**$_0$ Single-Step Diagram Lifting Algorithm)**.** *There is an algorithm* $\mathcal{A}^1_{\mathbb{C}}$ *which takes as input* $\langle \beta, \mathfrak{C}, p \rangle$ *s.t.*

- $\beta \in \mathbb{Z}[\vec{y}][x]^k$ *is a Muchnik sequence,*
- $\mathfrak{C}$ *is a* $(k \times m)$ *binary matrix (a* candidate *unlabeled extended root diagram for* β*),*
- $p \in \mathbb{Z}[\vec{y}][x]$ *is a non-constant polynomial w.r.t.* x *s.t.* $\beta^+ = \langle \beta(1), \ldots, \beta(k), p \rangle$ *is Muchnik*

and constructs a $(k+1 \times m')$ *binary matrix* $\mathfrak{C}^+$ *s.t.*

- if $\mathfrak{C}$ *is the unlabeled extended root diagram for* β *w.r.t.* $\vec{c} \in \mathbb{C}^n$*,* then $\mathfrak{C}^+$ *is the unlabeled extended root diagram for* β^+ *w.r.t. the same* $\vec{c}$*. That is,*

$$\mathfrak{C} = \mathcal{D}^*(\beta, \vec{c}) \implies \mathfrak{C}^+ = \mathcal{D}^*(\beta^+, \vec{c}).$$

As established by ***Lemma 2.1.26****, if* $\mathfrak{C}$ *is in fact not an unlabeled extended root diagram for* β *w.r.t. any* $\vec{c} \in \mathbb{C}^n$*, then it is of no consequence which* $(k+1 \times m')$ *binary matrix this algorithm returns. In certain cases, this algorithm may be able to "short-circuit" its processing by recognising that the candidate* $\mathfrak{C}$ *is not the unlabeled extended root diagram for* β *w.r.t. any* $\vec{c} \in \mathbb{C}^n$*. In these cases, the algorithm will return a special value* $\perp$ *to signify this.*

Proof. Let $deg(p) = d$ and $\alpha \in \mathbb{Z}[y_1, \ldots, y_n]$ be the highest degree coefficient of p (both w.r.t. x). Recall that as Muchnik sets are closed under partial differentiation, $d!\alpha$ appears in β and thus corresponds to a row in $\mathfrak{C}$. Let r be this row. If r is not a constant row, then $\mathfrak{C}$ cannot be an unlabeled extended root diagram for β, so we return $\perp$. Otherwise, we have two cases:

[Case I: $r = \vec{0}$] In this case, the root conditions for p are equivalent to those for $0 * x^d + \tau(p) = \tau(p) \in \mathbb{Z}[\vec{y}][x]$. But, note that $deg(\tau(p)) < d$ w.r.t. x. Thus, by definition of Muchnik sequence, we have that the row of roots for $\tau(p)$ already exists in $\mathfrak{C}$. Hence we may simply copy this row of roots as the row for p and we are done.

[Case II: $r = \vec{1}$] If $\mathfrak{C} = \mathcal{D}^*(\beta, \vec{c})$ for $\vec{c} \in \mathbb{C}^n$, then $r = \vec{1}$ yields that $\alpha(\vec{c}) \neq 0$. To extend $\mathfrak{C}$ to $\mathfrak{C}^+$ by taking into account p, we must meet the following requirements:

- Any root of p not already represented by a column of $\mathfrak{C}$ must be represented by a column of $\mathfrak{C}^+$ (columns must be added for these roots),

- The nullity of every polynomial represented by a row of $\mathfrak{C}$ at each new root of p must be determined,
- The nullity of p at all points represented by columns of $\mathfrak{C}$ must be determined (p will be 0 in every column of $\mathfrak{C}^+$ which is not present in $\mathfrak{C}$, as these are roots of p),
- The existence of an *anti-solution* column must be maintained.

Let ζ be a column of $\mathfrak{C}$. We have two cases:

[Case II.a: ζ is not an anti-solution column] So, the column ζ contains a 0 which does not come from a 0-row. Let $q \in \beta$ be of minimal degree ($deg(q) = e$) s.t. $\mathfrak{C}(q,\zeta) = 0$ and $\mathfrak{C}(q) \neq \vec{0}$. If $\mathfrak{C}$ is an extended root diagram for β w.r.t. $\vec{c} \in \mathbb{C}^n$, then this means that $q(\vec{c},\zeta) = 0$ and $q(\vec{c}) \in \mathbb{Z}[x]$ is not identically zero. Let $\gamma \in \mathbb{Z}[\vec{y}]$ be the highest degree coefficient of q s.t. $q = \gamma x^e + \tau(q)$. Observe by definition of Muchnik sequence that $e!\gamma$ corresponds to a row in $\mathfrak{C}$. If $\mathfrak{C}(e!\gamma)$ is not a constant row, then $\mathfrak{C}$ cannot be an unlabeled extended root diagram for β and we return $\perp$. Thus we assume $\mathfrak{C}(e!\gamma)$ is a constant row.

Let us now observe that if $\mathfrak{C}(e!\gamma) = \vec{0}$, then $\mathfrak{C}$ cannot be an unlabeled extended root diagram for β. If $\mathfrak{C}(e!\gamma) = \vec{0}$, then we have that the root conditions of q are equivalent to those of $0 + \tau(q) = \tau(q) \in \mathbb{Z}[\vec{y}][x]$. So, $\tau(q)(\vec{c},\zeta) = 0$. But then by assumption that q was of minimal degree with $q(\vec{c},\zeta) = 0$ and $\mathfrak{C}(q) \neq \vec{0}$, we have that $\mathfrak{C}(\tau(q))$ must be a 0-row. But, then $\mathfrak{C}(q)$ would be a 0-row as well, which is a contradiction. So, if $\mathfrak{C}(e!\gamma) = \vec{0}$ then $\mathfrak{C}$ cannot be an unlabeled extended root diagram for β and we return $\perp$. Thus we assume $\mathfrak{C}(e!\gamma) = \vec{1}$.

Let $r = rem(p,q)$ be the pseudo-remainder of p by q. So, $\gamma^{d-e+1}p = hq + r$ for some $h, r \in \mathbb{Z}[\vec{y}][x]$ s.t. $deg(r) \leq e-1$. As $\mathfrak{C}(q,\zeta) = 0$, if $\mathfrak{C}$ is an unlabeled extended root diagram for β w.r.t. $\vec{c} \in \mathbb{C}^n$, then $p(\vec{c}) = r(\vec{c})$. By definition of Muchnik sequence, r is represented by a row in $\mathfrak{C}$. Therefore we simply set $\mathfrak{C}^+(p,\zeta) = \mathfrak{C}(r,\zeta)$ and this case is complete. Observe that this process allows us to determine the nullity of p for every column of $\mathfrak{C}$ which is a root of some non-constant polynomial in β.

[Case II.b: ζ is an anti-solution column] So, the ζ only has 0's coming from 0-rows. We have two requirements left to meet:

- We must add columns to $\mathfrak{C}^+$ corresponding to the roots of p which are not represented by columns of $\mathfrak{C}$ and determine the nullity of each polynomial corresponding to a row of $\mathfrak{C}$ at these new roots of p,

- We must guarantee the existence of an anti-solution column for $\mathfrak{C}^+$.

As $p \in \mathbb{Z}[\vec{y}][x]$ is non-constant by assumption, we can extend ζ to be an anti-solution column of $\mathfrak{C}^+$ by simply setting $\mathfrak{C}^+(p,\zeta) = 1$. Thus, the requirements of both determining the nullity of p at every column label of $\mathfrak{C}$ and guaranteeing the existence of an anti-solution column for $\mathfrak{C}^+$ have been met.

It now remains to add columns to $\mathfrak{C}^+$ representing the roots of p not already represented by columns of $\mathfrak{C}$ and to determine the nullity of every polynomial represented by a row of $\mathfrak{C}$ at these new roots. Observe that any root of p not represented by a column of $\mathfrak{C}$ must have multiplicity 1, as otherwise it would be a root of $\frac{\partial p}{\partial x}$ and hence would be already represented by a column of $\mathfrak{C}$. By the Fundamental Theorem of Algebra, we may determine the number of new roots of p to add to $\mathfrak{C}^+$ as $(deg(p) - \#\kappa)$ where $\#\kappa$ is the number of roots of p already represented in $\mathfrak{C}$ counted with multiplicity. To determine $\#\kappa$, it will suffice to determine the multiplicity $m(\xi)$ of every root ξ of p appearing in $\mathfrak{C}$. To compute $m(\xi)$, we examine the successive derivatives $\frac{\partial p}{\partial x}, \frac{\partial^2 p}{\partial x^2}, \ldots$ and check the nullity of each derivative at ξ. By definition of Muchnik sequence, all such derivatives will correspond to rows of $\mathfrak{C}$, and thus this information may be computed from $\mathfrak{C}$. Then, $m(\xi) = j$ where j is the least power s.t. $\mathfrak{C}(\frac{\partial^j p}{\partial x^j}, \xi) = 1$. Thus, $\#\kappa = \sum_{\xi \in \chi} m(\xi)$ where χ is the collection of points represented by columns of $\mathfrak{C}$ s.t. $\mathfrak{C}(p,\xi) = 0$. Now, we add $(deg(p) - \#\kappa)$ new columns to $\mathfrak{C}^+$ with 0's in their bottom row (corresponding to p), 0's in their rows corresponding to 0-rows, and 1's in all other rows. As we have met our final requirements, this completes our proof. □

As it is easy to see all properties of $\mathbb{C}$ used in the above construction hold over every $\mathcal{F}$ s.t. $\mathcal{F} \models \mathbf{ACF}_0$, **Theorem 2.1.30** follows by induction along β. That is, we may always eliminate a single existential quantifier. From this result, **Theorem 2.1.9** follows by induction by placing $\mathcal{L}_\mathcal{R}$ formulas in prenex normal form and successively eliminating the innermost existential quantifier until no quantified variables remain.

2.2 Real Closed Fields

2.2.1 Ideals and Real Algebraic Varieties

Whereas classical algebraic geometry takes place over an algebraically closed field such as $\mathbb{C}$, real algebraic geometry takes place over a *real closed field* such as $\mathbb{R}$. The genesis of modern real algebraic geometry began most fervently with Artin-Schreier theory and Artin's subsequent solution to Hilbert's 17th Problem.

Question 2.2.1 (Hilbert's 17th Problem)**.** *Let $p \in \mathbb{R}[\vec{x}] = \mathbb{R}[x_1, \ldots, x_n]$ s.t. $p(\vec{r}) \geq 0$ for all $\vec{r} \in \mathbb{R}^n$. Does there then necessarily exist a representation of p as a sum of squares of finitely many real rational functions? That is, do there exist some $r_1, \ldots, r_k \in \mathbb{R}(\vec{x})$ s.t.*

$$p = \sum_{i=1}^{k} r_i^2$$

where $\mathbb{R}(\vec{x})$ is the field of real rational functions in $\vec{x}$?

Hilbert's 17th Problem was solved in the affirmative by Artin in 1926. Artin's approach can now be seen as being distinctively model-theoretic, and the theory both he and Schreier developed in the process of explicating the elementary structures sharing the ordered field properties of the real numbers introduced the notion of *real closed fields* and allowed real algebraic geometry to develop in a fairly general way (i.e., most core real algebraic constructions are given as taking place over *any* real closed field, not simply over $\mathbb{R}$).

2.2.2 Axiomatisation of RCF

We will give an elementary axiomatisation of the theory of real closed fields (**RCF**) as follows.

Definition 2.2.2 (Axiomatisation of **RCF**)**.** Let $\mathcal{L}$ be the first-order language with constants $\{0,1\}$, relation symbols $\{=,<\}$, function symbols $\{+,-,*\}$, and logical symbols $\{\wedge,\vee,\neg,\forall,\exists\}$. $\mathcal{L}$ is called the *language of ordered rings.* Then **RCF**, the elementary theory of real closed fields, is the $\mathcal{L}$-theory defined as follows:

$$\mathbf{RCF} = \mathbf{OF} \bigcup \mathbf{PSQR} \bigcup \mathbf{OP},$$

where

1. **OF** is an axiomatisation of the elementary $\mathcal{L}$-theory of *ordered fields*,
2. $\mathbf{PSQR} = \{\forall x \exists y (x < 0 \vee x = y * y\}$,
3. $\mathbf{OP} = \{\forall a_0, \ldots, a_{n-1} \exists x (a_0 + a_1 * x + a_2 * x^2 + \ldots + a_{2n+1} * x^{2n+1} = 0) \mid n \in \mathbb{N}\}$

Note that **OP**, the collection of $\mathcal{L}$-sentences stating that every odd degree univariate polynomial with coefficients in the field has a root in the field, is a countably infinite first-order axiom scheme.

The elementary theory **RCF** has many important properties. First and foremost, it is a *complete* theory. This groundbreaking result was proved by Tarski [Tar48]. Moreover, as **RCF** is recursively axiomatised, its completeness implies its decidability. In fact, as shown by Robinson [Rob56], **RCF** is *model-complete*. Most importantly for our work, **RCF** admits effective elimination of quantifiers. Let us now adapt the Muchnik quantifier elimination procedure given for $\mathbf{ACF}_0$ in **Section 2.1.3** to obtain a quantifier elimination procedure over **RCF**.

2.2.3 RCF Admits Quantifier Elimination

This section assumes the reader has mastered the analogous material over $\mathbb{C}$ presented in **Section 2.1.3**.

2.2.3.1 Real Sign Diagrams

Over $\mathbb{R}$, the analogue of complex rows of roots and root diagrams will be real *ordered rows of signs* and *sign diagrams*.

Definition 2.2.3 (Ordered Row of Signs)**.** Let $p \in \mathbb{Z}[x]$ with $\zeta_2 < \zeta_4 < \ldots < \zeta_{2m}$ its real roots, ignoring multiplicity. Observe that p has constant sign on the open intervals induced by the roots:

- $]-\infty, \zeta_2[$,
- $]\zeta_{2i}, \zeta_{2i+2}[$ $(\forall 1 \leq i < m)$, and
- $]\zeta_{2m}, +\infty[$.

Given a pair of consecutive roots ζ_{2i}, ζ_{2i+2}, let ζ_{2i+1} be any point in $]\zeta_{2i}, \zeta_{2i+2}[$. Similarly, let ζ_1 be any point in $]-\infty, \zeta_2[$ and ζ_{2k+1} be any point in $]\zeta_{2k}, +\infty[$ (we will see in **Chapter 6** that choosing a *rational point* within the open intervals can be advantageous, though for now this does not matter). All such points ζ_j (including the roots) are called *sample points*. Then, an *ordered row of signs* for p is simply the sequence of signs of p at each ζ_j, e.g.,

$$\langle sgn(p(\zeta_1), sgn(p(\zeta_2)), \ldots, sgn(p(\zeta_{2m+1}))\rangle,$$

where

$$sgn(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

We write $\mathcal{R}_{\mathbb{R}}(\alpha, p)$ to mean that α is an ordered row of signs for p.

We must now extend our single polynomial machinery so as to handle a set of polynomials. In the set case, roots of polynomials will not necessarily appear as the sample points with even index. This leads naturally to the definition of a *sign diagram*.

Definition 2.2.4 (Sign diagram)**.** Let $P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[x]$. Let M be a $(k \times m)$ matrix with entries in $\{-1, 0, 1\}$. M will be a *sign diagram* for P if there exists a sequence of sample points $\Theta = \langle \zeta_1, \ldots, \zeta_m \rangle \in \mathbb{R}^m$ with $\zeta_1 < \zeta_2 < \ldots < \zeta_m$ s.t.

- $\forall p_i \neq 0 \in P \, \forall r \in \mathbb{R} \left(p_i(r) = 0 \implies \exists \zeta_j \in \Theta(\zeta_j = r) \right)$,
- $\forall \zeta_i \neq \zeta_j \in \Theta[(\exists p_u \neq 0, p_v \neq 0 \in P(p_u(\zeta_i) = 0 \ \wedge \ p_v(\zeta_j) = 0))$
  $\implies \exists \zeta_w \in \Theta(\zeta_i < \zeta_w < \zeta_j \ \wedge \ \forall p_s \in P(p_s(\zeta_w) \neq 0)]$,
- $\forall p_i \in P \, \forall \zeta_j \in \Theta \left(M(p_i, \zeta_j) = sgn(p_i(\zeta_j)) \right)$,
- $\forall p_i \neq 0 \in P \left(p_i(\zeta_1) \neq 0 \ \wedge \ p_i(\zeta_m) \neq 0 \right)$,

where members of P have been used to refer the rows of M and members of Θ to refer its columns: $M(p_i, \zeta_j)$ is the $\langle i, j \rangle$-th entry of M, $M(p_i)$ is the j-th row of M and $M(\zeta_j)$ is the j-th column of M. If M is a sign diagram for P as above, then we will call Θ a *witnessing sample point sequence* for M.

Observation 2.2.5. If M is a sign diagram for P, then $M(p_i)$ is *essentially* an ordered row of signs for p_i, with the caveat that from the perspective of p_i, $M(p_i)$ may contain some duplicate entries. These duplicate entries correspond to extra sample points in the intervals between roots of p_i, and may be roots of other $p_j \in P$, or sample points in between them.

In the following, let $M \in \{-1, 0, 1\}^{k \times m}$ be a sign diagram for $P = \{p_1, \ldots, p_k\}$ with $\Theta = \langle \zeta_1, \ldots, \zeta_m \rangle$ a witnessing sample point sequence for M.

Observation 2.2.6. If $p_i \in P$ is identically 0, then $M(p_i) = \vec{0}$.

Definition 2.2.7 (Root of P)**.** A sample point ζ_j which is a root of some $p_i \in P$ is called *a root of P*.

Observation 2.2.8. The collection of *roots of P* is the same as the collection of roots of the single polynomial $\prod_{p_i \in P} p_i$.

Definition 2.2.9 (Non-root of P)**.** A sample point ζ_j which is not a root of any $p_i \in P$ is called *a non-root of P*.

Observation 2.2.10. Two adjacent columns $M(\zeta_i)$ and $M(\zeta_{i+1})$ of M cannot be equal unless both ζ_i and ζ_{i+1} are non-roots of P.

Observation 2.2.11. If two adjacent columns of M are equal and M' is obtain from M by removing one of the duplicate columns, then M' is again a sign diagram for P.

Definition 2.2.12 (Minimal sign diagram)**.** A *minimal sign diagram for P* is a sign diagram M for P s.t. M does not contain any duplicate adjacent columns. In particular, M consists of

- an initial column corresponding to a sample point ζ_1 which is less than any root of P,
- sequences of adjacent columns corresponding to sample points of the form $\zeta_j < \zeta_{j+1} < \zeta_{j+2}$ s.t. ζ_j and ζ_{j+2} are roots of P and ζ_{j+1} is a non-root of P, and
- a final column corresponding to a sample point ζ_m which is greater than any root of P.

There is then an analogue of **Lemma 2.1.15** for sign diagrams, whose proof is immediate:

Lemma 2.2.13 (Uniqueness of Minimal Sign Diagrams)**.** *Let M, M' be minimal sign diagrams for P. Then, $M = M'$.*

Observation 2.2.14. If M is a sign diagram for P, then for any $r \in \mathbb{R}$, we may find a column $M(\zeta_j)$ of M s.t. $\forall p_i \in P(M(p_i, \zeta_j) = sgn(p_i(r)))$.

Observation 2.2.15. If M is a sign diagram for P, then there can never be two consecutive columns $M(\zeta_j), M(\zeta_{j+1})$ s.t. $M(p_i, \zeta_j) = M(p_i, \zeta_{j+1}) = 0$ for any $p_i \in P$, unless $M(p_i) = \vec{0}$. That is, there can never be two consecutive 0's in a row of M unless they appear in a row consisting only of 0's (corresponding to $p_i \in P$ which is identically 0).

2.2.3.2 Elimination of a Single Existential Quantifier

With our adjusted algebraic machinery in hand, the proof of quantifier elimination over **RCF** will go through identically as it did for $\mathbf{ACF}_0$, except for the construction of the *diagram lifting algorithm*. It will thus suffice to present the real adapation ($\mathcal{A}_{\mathbb{R}}$) of the

complex diagram lifting algorithm ($\mathcal{A}_{\mathbb{C}}$). As with the complex case, $\mathcal{A}_{\mathbb{R}}$ will exploit the inductive nature of Muchnik sequences by recursively making use of a "single-step lifting algorithm," $\mathcal{A}^1_{\mathbb{R}}$. Once we have exhibited such an algorithm, we will have proved the following theorem establishing, by induction, that **RCF** admits elimination of quantifiers.

Theorem 2.2.16 (Projective Closure of Definability)**.** *Given any quantifier-free $\mathcal{L}$-formula $\varphi(\vec{y},x)$ there exists a quantifier-free $\mathcal{L}$-formula $\psi(\vec{y})$ s.t.*

$$\boldsymbol{RCF} \models \forall \vec{y}\ (\exists x \varphi(\vec{y},x) \iff \psi(\vec{y}))\,.$$

Moreover, $\psi(\vec{y})$ is effectively computable from $\varphi(\vec{y},x)$.

Observe that unlike root diagrams in the complex case, sign diagrams are by definition simply matrices and do not carry explicit row nor column labels. We will use the notion of an *extended sign diagram* to be the real analogue of a complex *unlabeled extended root diagram* (**Definition 2.1.28**) in the obvious way. Given a finite set of polynomials $S \subset \mathbb{Z}[\vec{y}][x]$ and $\vec{r} \in \mathbb{R}^n$, we will write $\mathcal{D}^*(S,\vec{r})$ to mean the minimal extended sign diagram for S w.r.t. $\vec{r}$. Similarly, if β is a Muchnik sequence of polynomials in $\mathbb{Z}[\vec{y}][x]$, then $\mathcal{D}^*(\beta,\vec{r})$ will be the extended sign diagram for the underlying set of β w.r.t. $\vec{r}$.

Recall that by **Lemma 2.1.26**, every subsequence β' of β is Muchnik. Given an unlabeled extended root diagram for β_0 w.r.t. $\vec{r}$, we will use this property of Muchnik sequences to build an extended sign diagram for β w.r.t. $\vec{r}$ inductively, by building one for each of its subsequences β'.

Lemma 2.2.17 (**RCF** Diagram Lifting Algorithm)**.** *There is an algorithm $\mathcal{A}^1_{\mathbb{R}}$ which takes as input $\langle \beta, \mathfrak{C}, p\rangle$ s.t.*

- *$\beta \in \mathbb{Z}[\vec{y}][x]^k$ is a Muchnik sequence,*
- *$\mathfrak{C}$ is a $(k \times m)$ binary matrix (a* candidate *minimal extended sign diagram for β),*
- *$p \in \mathbb{Z}[\vec{y}][x]$ is a non-constant polynomial w.r.t. x s.t. $\beta^+ = \langle \beta(1),\ldots,\beta(k),p\rangle$ is Muchnik*

and constructs a $(k+1 \times m')$ binary matrix $\mathfrak{C}^+$ s.t.

- if $\mathfrak{C}$ *is the minimal extended sign diagram for β w.r.t. $\vec{r} \in \mathbb{R}^n$,* then $\mathfrak{C}^+$ *is the minimal extended sign diagram for β^+ w.r.t. the same $\vec{r}$. That is,*

$$\mathfrak{C} = \mathcal{D}^*(\beta,\vec{r}) \implies \mathfrak{C}^+ = \mathcal{D}^*(\beta^+,\vec{r}).$$

It follows by the real analogue of ***Lemma 2.1.26*** *that if* $\mathfrak{C}$ *is in fact not a minimal extended sign diagram for* β *w.r.t. any* $\vec{r} \in \mathbb{R}^n$*, then it is of no consequence which* $(k+1 \times m')$ *binary matrix this algorithm returns. In certain cases, this algorithm may be able to "short-circuit" its processing by recognising that the candidate* $\mathfrak{C}$ *is not the minimal extended sign diagram for* β *w.r.t. any* $\vec{r} \in \mathbb{R}^n$*. In these cases, the algorithm will return a special value* $\perp$ *to signify this.*

Proof. Assume that $\mathfrak{C}$ is the minimal extended sign diagram for β w.r.t. $\vec{r} \in \mathbb{R}^n$. Thus, there exists a witnessing sample point sequence for M, $\Theta = \langle \zeta_1, \ldots, \zeta_m \rangle$. Let $deg(p) = d$ and $\alpha \in \mathbb{Z}[y_1, \ldots, y_n]$ be the highest degree coefficient of p (both w.r.t. x). Recall that as Muchnik sets are closed under partial differentiation, $d!\alpha$ appears in β and thus corresponds to a row in $\mathfrak{C}$. Let r be this row. If r is not a constant row, then $\mathfrak{C}$ cannot be an extended sign diagram for β, so we return $\perp$. Otherwise, we have two cases:

[Case I: $r = \vec{0}$] In this case, the sign conditions for p are equivalent to those for $0 * x^d + \tau(p) = \tau(p) \in \mathbb{Z}[\vec{y}][x]$. But, note that $deg(\tau(p)) < d$ w.r.t. x. Thus, by definition of Muchnik sequence, we have that the row of signs for $\tau(p)$ already exists in $\mathfrak{C}$. Hence we may simply copy this row of signs as the row for p and we are done.

[Case II: $r \neq \vec{0}$] We must incorporate the sign conditions of $p(\vec{r}) \in \mathbb{R}[x]$ into $\mathfrak{C}^+$. In obtaining $\mathfrak{C}^+$ from $\mathfrak{C}$, this requires we accomplish the following goals:

- Columns must be added corresponding to roots of $p(\vec{r})$ which do not correspond to any columns in $\mathfrak{C}$ (i.e., columns must be added corresponding to roots of $p(\vec{r})$ which do not already appear in Θ),
- Columns must be added corresponding to sample points in between roots and before and after the first and last roots (if these change in the process of adding columns corresponding to the roots of $p(\vec{r})$),
- The sign of $p(\vec{r})$ at the sample point ζ_i corresponding to each column must be determined,
- The sign of all other polynomials in β (i.e., those corresponding to the rows of $\mathfrak{C}$), must be determined for all new columns which are added.

We have a number of subcases to consider. Let ζ_i be a sample point corresponding to a column of $\mathfrak{C}$. Note that in all of the reasoning below, we never have to determine a value for the sample point ζ_i — all of the information we need about ζ_i is contained in the column corresponding to it.

[Case II.a: $\zeta_i = \zeta_1$ or $\zeta_i = \zeta_m$] In this case, we are determining the sign of p at either the first or last column of $\mathfrak{C}$. Recall that a polynomial is eventually (w.r.t. the absolute value of its input) dominated by its highest degree monomial. Thus, $\mathfrak{C}^+(p,\zeta_i)$ should be set to the sign of $p(\vec{r})$ at either $-\infty$ or $+\infty$, respectively. By definition of Muchnik sequence, the leading coefficient α of p w.r.t. x corresponds to a (constant) row of $\mathfrak{C}$, and so we know its sign at $\vec{r}$: $sgn(\alpha(\vec{r})) = \mathfrak{C}(\alpha,\zeta_i)$. Thus, we set $\mathfrak{C}^+(p,\zeta_i)$ to either $(-1)^d\mathfrak{C}(\alpha,\zeta_i)$ or $\mathfrak{C}(\alpha,\zeta_i)$.

[Case II.b: ζ_i is a root of a non-zero polynomial in β] Then, there is some $q \in \beta$ s.t. $\mathfrak{C}(q,\zeta_i) = 0$ and $\mathfrak{C}(q) \neq \vec{0}$. Let $q \in \beta$ be of minimal degree ($deg(q) = e$) s.t. $\mathfrak{C}(q,\zeta_i) = 0$ and $\mathfrak{C}(q) \neq \vec{0}$. By assumption that $\mathfrak{C}$ is the minimal extended sign diagram for β w.r.t. $\vec{r} \in \mathbb{R}^n$, it follows that $q(\vec{r},\zeta_i) = 0$ and $q(\vec{r}) \in \mathbb{Z}[x]$ is not identically zero. Let $\gamma \in \mathbb{Z}[\vec{y}]$ be the highest degree coefficient of q s.t. $q = \gamma x^e + \tau(q)$. Observe by definition of Muchnik sequence that $e!\gamma$ corresponds to a row in $\mathfrak{C}$. If $\mathfrak{C}(e!\gamma)$ is not a constant row, then $\mathfrak{C}$ cannot be an extended sign diagram for β and we return $\perp$. Thus we assume $\mathfrak{C}(e!\gamma)$ is a constant row.

Let us now observe that if $\mathfrak{C}(e!\gamma) = \vec{0}$, then $\mathfrak{C}$ cannot be an extended sign diagram for β. If $\mathfrak{C}(e!\gamma) = \vec{0}$, then we have that the sign conditions of q are equivalent to those of $0 + \tau(q) = \tau(q) \in \mathbb{Z}[\vec{y}][x]$. So, $\tau(q)(\vec{r},\zeta) = 0$. But then by assumption that q was of minimal degree with $q(\vec{r},\zeta) = 0$ and $\mathfrak{C}(q) \neq \vec{0}$, we have that $\mathfrak{C}(\tau(q))$ must be a 0-row. But, then $\mathfrak{C}(q)$ would be a 0-row as well, which is a contradiction. So, if $\mathfrak{C}(e!\gamma) = \vec{0}$ then $\mathfrak{C}$ cannot be an extended sign diagram for β and we return $\perp$. Thus we assume $\mathfrak{C}(e!\gamma) = \vec{1}$.

Let $r = rem(p,q)$ be the pseudo-remainder of p by q. So, $\gamma^{d-e+1}p = hq + r$ for some $h, r \in \mathbb{Z}[\vec{y}][x]$ s.t. $deg(r) \leq e-1$. As $\mathfrak{C}(q,\zeta) = 0$, for $\mathfrak{C}$ to be an extended sign diagram for β w.r.t. $\vec{r} \in \mathbb{R}^n$, then we must have $p(\vec{r}) = r(\vec{r})$. By definition of Muchnik sequence, $r \in \beta$ so r corresponds to a row in $\mathfrak{C}$. Therefore we simply set $\mathfrak{C}^+(p,\zeta_i) = \mathfrak{C}(r,\zeta_i)$ and this case is complete. Observe that this process allows us to determine the nullity of p at every column corresponding to a sample point ζ_i which is a root of some non-constant polynomial in β.

[Case II.c: ζ_i is not the root of non-constant polynomial in β] As the first and last columns have already been handled, it follows that ζ_i must be a sample point between two roots of non-constant polynomials in β. Let ζ_- and ζ_+ be these two roots (which correspond to the columns immediately to the left and right of the column represented by ζ_i). By the argument in the previous case, we have already determined the sign of p at these two roots as $\mathfrak{C}^+(p,\zeta_-)$ and $\mathfrak{C}^+(p,\zeta_+)$. Let $\varepsilon_- = \mathfrak{C}^+(p,\zeta_-)$ and

$\varepsilon_+ = \mathfrak{C}^+(p, \zeta_+)$. Let us now consider the possible values of ε_- and ε_+ below.

[Case II.c.i: $(\varepsilon_- = \varepsilon_+ = 0)$] By Rolle's Theorem, which holds over every real closed field, there must be some $\eta \in]\zeta_-, \zeta_+[$ s.t. $\frac{\partial p}{\partial x}(\vec{r}, \eta) = 0$. By the definition of Muchnik sequence, β contains $\frac{\partial p}{\partial x}$ and the assumption that $\mathfrak{C}$ is the minimal extended sign diagram for β w.r.t. $\vec{r}$ yields that there is a column of $\mathfrak{C}$ corresponding to η. Thus, $\eta = \zeta_i$. But, then $\mathfrak{C}(\frac{\partial p}{\partial x})$ must be $\vec{0}$, and so $\frac{\partial p}{\partial x}(\vec{r}) \in \mathbb{R}[x]$ must be identically 0. But, if $\frac{\partial p}{\partial x}(\vec{r})$ is identically 0, then $p(\vec{r}) \in \mathbb{R}[x]$ must be a constant function, and since it obtains 0 by assumption $\varepsilon_- = \varepsilon_+ = 0$, it must be identically 0 as well. But, then the sign conditions for p should have already been handled by Case I. Since this did not happen, this means the row corresponding to $d!\alpha \neq \vec{0}$. But, this is a contradiction. So, if $\varepsilon_- = \varepsilon_+ = 0$, then we return $\perp$.

[Case II.c.ii: $\neg(\varepsilon_- = \varepsilon_+ = 0)$] First, assume ε_- and ε_+ have opposite non-zero signs. By the Intermediate Value Theorem, which holds over every real closed field, there must be some $\eta \in]\zeta_-, \zeta_+[$ s.t. $p(\vec{r}, \eta) = 0$. There is no guarantee that η coincides with ζ_i. By assumption that $\mathfrak{C}$ is the extended sign diagram for β w.r.t. $\vec{r}$, it follows that no other row of $\mathfrak{C}^+$ will change its behavior in the interval $]\zeta_-, \zeta_+[$. Thus, we replace the column $\mathfrak{C}(\zeta_i)$ with three copies of itself and extend them to account for p in $\mathfrak{C}^+$ by setting their final row entries to be ε_-, 0, and ε_+, respectively.

Finally, assume that either exactly one of ε_-, ε_+ is 0 or $\varepsilon_- = \varepsilon_+ \neq 0$. The work we must do for both of these cases will be identical, as both cases yield that $p(\vec{r})$ must have no root in $]\zeta_-, \zeta_+[$. For suppose there were such a root $\eta \in]\zeta_-, \zeta_+[$ s.t. $p(\vec{r}, \eta) = 0$. It will follow that $\frac{\partial p}{\partial x}(\vec{r})$ must have a root in $]\zeta_-, \zeta_+[$ which we have seen leads to a contradiction by the argument in Case II.c.i. First, suppose that exactly one of the ε_-, ε_+ is 0. Without loss of generality, say $\varepsilon_- = 0$. Then, we have $\zeta_- < \eta < \zeta_+$ s.t. both ζ_- and η are roots of $p(\vec{r})$. But then by Rolle's Theorem, it follows that $\frac{\partial p}{\partial x}(\vec{r})$ has a root in $]\zeta_-, \eta[$ as desired. On the other hand, suppose $\varepsilon_- = \varepsilon_+ = s$ with $s \in \{-1, 1\}$. Then, for $p(\vec{r})$ to have a root in $]\zeta_-, \zeta_+[$, $\frac{\partial p}{\partial x}(\vec{r})$ would have to change sign in $]\zeta_-, \zeta_+[$, and hence by the Intermediate Value Theorem, it follows that $\frac{\partial p}{\partial x}(\vec{r})$ would have a root in $]\zeta_-, \zeta_+[$. So, in both cases, the assumption that $p(\vec{r})$ has a root in $]\zeta_-, \zeta_+[$ leads to a contradiction. Thus, the sign of $p(\vec{r})$ on $]\zeta_-, \zeta_+[$ does not change, and is then equal to the sign at the endpoint which is not a root of $p(\vec{r})$. So, we set $\mathfrak{C}^+(p, \zeta_i)$ to the sign ε_- or ε_+ which is non-zero. As we have met our final requirements, this completes our proof. □

2.3 Gröbner Bases

In this section, we will present an overview of Gröbner basis theory. In the process, we fix notation which will then be assumed and used freely throughout the rest of this dissertation. Though our exposition is original, we have benefited greatly from the careful presentations in [CLO07] and [Has07]. Unlike these books, we emphasise the *rewriting* perspective of Gröbner bases, as it is perhaps the most intuitive one for our intended audience.

2.3.1 An Intuitive Sketch

Gröbner bases are a fundamental tool for solving algorithmic problems in classical algebraic geometry. In what follows, let $\mathcal{I} = \mathcal{I}(\{b_1, \ldots, b_k\})$ be an ideal of $\mathbb{Q}[\vec{x}]$ and let $p \in \mathbb{Q}[\vec{x}]$. Some important problems Gröbner bases will allow us to solve include:

- Ideal membership: Is $p \in \mathcal{I}$?
- Canonicalisation in quotient rings: What is a canonical choice of representative for the equivalence class of p in the quotient ring $\mathbb{Q}[\vec{x}]/\mathcal{I}$?
- Complex satisfiability: Does the equational system induced by $\mathcal{I}$, $(\wedge_{i=1}^{k} b_i = 0)$, have a solution over $\mathbb{C}^n$? Equivalently, is $\mathcal{V}_{\mathbb{C}}(\mathcal{I}) = \emptyset$? Also, equivalently: does $\mathbf{ACF}_0 \models \exists \vec{x}(\wedge_{i=1}^{k} b_i = 0)$?
- Ideal equality: Given another ideal $\mathcal{J} = \mathcal{I}(\{c_1, \ldots, c_j\})$, is $\mathcal{I} = \mathcal{J}$?
- Dimension of ideal: If $\mathcal{V}_{\mathbb{C}}(\mathcal{I}) \neq \emptyset$, then is $|\mathcal{V}_{\mathbb{C}}(\mathcal{I})| < \omega$? Equivalently, is $\mathcal{I}$ zero-dimensional?
- Complex triangulation: If $|\mathcal{V}_{\mathbb{C}}(\mathcal{I})| < \omega$, then what are its members, presented as a triangulated system of equations?

What then is a Gröbner basis? Given an ideal such as $\mathcal{I} = \mathcal{I}(\{b_1, \ldots, b_k\})$, recall that $\{b_1, \ldots, b_k\}$ is called a *basis* for $\mathcal{I}$. Hilbert's Basis Theorem (**Theorem 2.1.5**) guarantees that all ideals over $\mathbb{Q}[\vec{x}]$ are finitely generated in this way. A Gröbner basis for $\mathcal{I}$ is a special type of basis $\{g_1, \ldots, g_j\}$ for $\mathcal{I}$ (e.g., $\mathcal{I} = \mathcal{I}(\{b_1, \ldots, b_k\}) = \mathcal{I}(\{g_1, \ldots, g_j\})$) whose polynomials g_i have a syntactic structure which will allow us to solve all of the above problems systematically.

Intuitively, Gröbner basis theory is built on the following observations:

- A special type of *well-founded, total ordering* $\prec$ may be placed upon the monomials of $\mathbb{Q}[\vec{x}]$ so that every polynomial in $\mathbb{Q}[\vec{x}]$ has a *leading monomial* (often called a *leading term* or *head term*) w.r.t. $\prec$. To gain intuition, let us imagine that we have fixed such an order $\prec$. Given a polynomial $p = cm + q \in \mathbb{Q}[\vec{x}]$, we will write $p = \underline{cm} + q$ to mean that cm is the leading monomial of p w.r.t. $\prec$.

- Recall that as an ideal is a algebraic generalisation of *nullity* – e.g., ideals are the *kernels* of ring homomorphisms $\mathbb{Q}[\vec{x}] \mapsto \mathbb{Q}[\vec{x}]/\mathcal{I}$ – we can interpret $p \in \mathcal{I}$ as meaning that *from the perspective of the quotient ring* $\mathbb{Q}[\vec{x}]/\mathcal{I}$, $p = 0$.

- Thus, if $p = \underline{cm} + q \in \mathcal{I}$, we may associate with p an equation $\underline{cm} + q = 0$ and thus also a *rewrite rule*:
$$m \to (\frac{1}{c})(-q).$$

- It is easy to see that this view of a basis $\{b_1, \ldots, b_k\}$ of $\mathcal{I}$ as providing a Noetherian system of rewrite rules ($\mathcal{R}$) has the following property:
$$p \xrightarrow{\mathcal{R}} q \implies (p - q) \in \mathcal{I}.$$
Stated another way,
$$p \xrightarrow{\mathcal{R}} q \implies [p] = [q],$$
where $[p]$ is the equivalence class of p in $\mathbb{Q}[\vec{x}]/\mathcal{I}$. Equivalently, given the ring homomorphism induced by $\mathcal{I}$ as $h : \mathbb{Q}[\vec{x}] \to \mathbb{Q}[\vec{x}]/\mathcal{I}$,
$$p \xrightarrow{\mathcal{R}} q \implies h(p) = h(q).$$
Thus, given any $p \in \mathbb{Q}[\vec{x}]$, we will have a sufficient condition for *ideal membership*:
$$p \xrightarrow{\mathcal{R}} 0 \implies p \in \mathcal{I}.$$

- But, this condition need not be *necessary*. The rewrite system $\mathcal{R}$ may have the property that some p is not reduced to 0 even though $p \in \mathcal{I}$. The obstruction is that $\mathcal{R}$ is not guaranteed to be *confluent*. A *Gröbner basis* for $\mathcal{I}$ w.r.t. $\prec$ is a basis s.t. the induced rewrite system *is* confluent, and thus all of the sufficient conditions above become necessary as well.

Let us now make the above intuitive sketch precise.

2.3.2 Foundations of Gröbner Basis Theory

We use $\mathbb{Q}[\vec{x}]$ to denote the polynomial ring $\mathbb{Q}[x_1,\ldots,x_n]$. A function $\alpha \in \mathbb{N}^n$ is called an *exponent vector*. Given a sequence of indeterminates $x_1,\ldots,x_n$ and an exponent vector $\alpha \in \mathbb{N}^n$, a *power-product* is a formal product of the form $x_1^{\alpha(1)}\ldots x_n^{\alpha(n)}$. When no ambiguity can arise, we will write $x_1^{\alpha(1)}\ldots x_n^{\alpha(n)}$ as $\vec{x}^{\alpha}$. An element $c\vec{x}^{\alpha}$ with $c \in \mathbb{Q}$ and $\vec{x}^{\alpha}$ a power-product is called a *monomial*. We say a monomial is *monic* if $c = 1$. A *polynomial* is a finite sum of monomials of the form

$$\sum_{\alpha\in\mathcal{E}} c_\alpha \vec{x}^{\alpha} \quad \text{with} \quad \mathcal{E} \subset \mathbb{N}^n \quad \text{and} \quad c_\alpha \in \mathbb{Q}\setminus\{0\}.$$

An important property to note is that as formal objects, such polynomials have monomial summands with equal power-products combined by summing their coefficients. This representation is called *sparse sum-of-monomials normal form* and will be from now on assumed.

We use $\mathbb{M}$ to denote the set of all power-products in $\mathbb{Q}[\vec{x}]$. We use p, q and r to denote polynomials, m to denote power-products and monic monomials, c to denote coefficients, and cm to denote monomials. All such symbols may be subscripted.

We say a power-product $\vec{x}^{\alpha}$ *contains* x_k if $\alpha(k) > 0$. Given two power-products $m_1 = x_1^{i_1}\ldots x_n^{i_n}$ and $m_2 = x_1^{j_1}\ldots x_n^{j_n}$, m_1m_2 denotes the power-product $x_1^{i_1+j_1}\ldots x_n^{i_n+j_n}$. If $i_k \geq j_k$ for $k \in \{1,\ldots,n\}$, then $\frac{m_1}{m_2}$ denotes the power-product $x_1^{i_1-j_1}\ldots x_n^{i_n-j_n}$. The *least common multiple* $\mathsf{lcm}(m_1,m_2)$ of m_1 and m_2 is $x_1^{max(i_1,j_1)}\ldots x_n^{max(i_n,j_n)}$.

An ordering relation $\prec$ on the set $\mathbb{M}$ is *admissible* if $m_1 \prec m_2$ implies that $m_1m \prec m_2m$, for all m_1, m_2 and m in $\mathbb{M}$. A *monomial order* is a total order on $\mathbb{M}$ which is admissible and a well ordering.

Let us make the above definitions concrete by presenting two common monomial orders.

Example 2.3.1 (Lexicographic order)**.** *The* lexicographic order $\prec_{lex}$ *is defined as*

$$x_1^{i_1}\ldots x_n^{i_n} \prec_{lex} x_1^{j_1}\ldots x_n^{j_n}$$

$$\Longleftrightarrow$$

$$\exists 0 \leq k < n \;\; s.t. \;\; i_1 = j_1 \wedge \ldots \wedge i_k = j_k \wedge i_{k+1} < j_{k+1}.$$

Example 2.3.2 (Degree-reverse lexicographic order)**.** *The* degree reverse lexicographic order $\prec_{dlex}$ *is defined as*

$$M_1 = x_1^{i_1}\ldots x_n^{i_n} \prec_{dlex} x_1^{j_1}\ldots x_n^{j_n} = M_2$$

$$\Longleftrightarrow$$

$$deg(M_1) < deg(M_2)$$

$$\vee$$

$$[deg(M_1) = deg(M_2) \wedge (\exists 1 < k \leq n+1 \ \ s.t. \ \ i_n = j_n \wedge \ldots \wedge i_k = j_k \wedge i_{k-1} > j_{k-1})].$$

Given a monomial order $\prec$, we can lift it to an order on polynomials. Given two polynomials p_1 and p_2, we say $p_1 \prec p_2$ if there exists a power-product m s.t.

- m is contained in p_2 and not contained in p_1, and
- $\forall\, m'$ s.t. $m \prec m'$ we have m' contained in p_1 iff m' contained in p_2.

From now on, we often assume a monomial order $\prec$ has been chosen. When we write a polynomial p as

$$\underline{cm} + q,$$

we mean that cm is the head monomial of p w.r.t. the background monomial order, and m is not contained in q.

Definition 2.3.3 (Monic polynomial)**.** A polynomial $\underline{cm} + q$ is *monic* if $c = 1$.

Definition 2.3.4 (S-polynomial)**.** Given two monic polynomials p_1 and p_2 of the form $\underline{m_1} + q_1$ and $\underline{m_2} + q_2$, let $\tau_{1,2}$ be the $\mathsf{lcm}(m_1, m_2)$, then we use $\mathsf{spol}(p_1, p_2)$ to denote the polynomial

$$(\frac{\tau_{1,2}}{m_1})q_1 - (\frac{\tau_{1,2}}{m_2})q_2.$$

Observation 2.3.5. Given a set of polynomials S, it is easy to see that if $\{p_1, p_2\} \subseteq \mathcal{I}(S)$, then $\mathsf{spol}(p_1, p_2) \in \mathcal{I}(S)$.

Given a monomial order $\prec$, a key idea underlying Gröbner bases is to use a polynomial $\underline{cm} + q$ as a rewrite rule $m \to (\frac{1}{c})(-q)$.

To simplify the presentation that follows, we will assume all polynomials used as rewrite rules are monic. This is no restriction, as any set of polynomials can be made monic (by dividing through by the head rational coefficient of the head monomial) without altering the ideal it generates. The monic polynomial $p = \underline{m} + q$ induces a *reduction relation* $\mapsto_p$ on polynomials as follows.

Definition 2.3.6 (Reduction relation of monic polynomial)**.** The reduction relation $\mapsto_p$ induced by a monic polynomial

$$p = \underline{m} + q$$

is defined as

$$q_1 + c_1 m_1 m \;\; \mapsto_p \;\; q_1 - c_1 m_1 q$$

for arbitrary monomials $c_1 m_1$ and polynomials q_1 not containing $m_1 m$.

Similarly, a set of monic polynomials induces a reduction relation by suitably combining the reduction relation of its members.

Definition 2.3.7 (Reduction relation of set of monic polynomials)**.** Given a set of monic polynomials $G = \{p_1, \ldots, p_k\}$, the reduction relation $\mapsto_G$ induced by G is defined as

$$\mapsto_G \;=\; \bigcup_{i=1}^{k} \mapsto_{p_i} .$$

Observation 2.3.8 (Reduction relation is Noetherian)**.** Observe that given a monomial order $\prec$, the reduction relation induced by a set of monic polynomials $G = \{p_1, \ldots, p_k\}$ is Noetherian.

The fact that our reduction relations are Noetherian allows us to introduce the notion of a *residue* of a polynomial w.r.t. a set of monic polynomials.

Definition 2.3.9. Given a set of monic polynomials $G = \{p_1, \ldots, p_k\}$ and a polynomial p, a residue of p w.r.t. G is defined as a polynomial q s.t.

$$p \mapsto_G q_1 \mapsto_G q_2 \mapsto_G \cdots \mapsto_G q \quad \text{but} \quad \neg\exists r(q \mapsto_G r).$$

We will write

$$p \xrightarrow{G} q$$

to mean that q is a residue of p w.r.t. G.

Observation 2.3.10 (Residue is in ideal)**.** Observe that if $p \xrightarrow{G} q$, then $(p - q) \in \mathcal{I}(G)$. In particular, if $p \xrightarrow{G} 0$, then $p \in \mathcal{I}(G)$. Moreover, if $p \in \mathcal{I}(G)$ and $p \xrightarrow{G} q$, then $q \in \mathcal{I}(G)$.

Observation 2.3.11 (Equivalence of residue uniqueness and confluence)**.** Observe that all polynomial residues w.r.t. G are uniquely defined if and only if $\mapsto_G$ is confluent.

Input: $\langle F = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\vec{x}], \prec \rangle$
Output: G s.t. G is a Gröbner basis of F w.r.t. $\prec$
$G := F$
repeat
  $G' := G$
  for each pair $\langle p_i, p_j \rangle \in (G' \times G')$ with $p_i \neq p_j$ **do**
    Let q be s.t. $spol(p_i, p_j) \xrightarrow{G'} q$
    if $q \neq 0$ **then**
      $G := G \cup \{q\}$
    end if
  end for
until $G = G'$

Figure 2.1: Buchberger's Algorithm

At last, we may define the concept of a Gröbner basis. We restrict ourselves to monic bases as every Gröbner basis for an ideal $\mathcal{I} \subset \mathbb{Q}[\vec{x}]$ may be trivially made monic, and this restriction simplifies our exposition in subsequent chapters.

Definition 2.3.12 (Gröbner basis)**.** A finite set of monic polynomials G is a Gröbner basis of the ideal $\mathcal{I}(F)$ iff

$$\mathcal{I}(G) = \mathcal{I}(F) \quad \text{and} \quad \mapsto_G \text{ is confluent.}$$

2.3.3 Classical Basis Construction Algorithms

The first algorithm for Gröbner basis construction was given by Buchberger in his PhD thesis [Buc65]. It has since been optimised in many ways. Before touching on these optimisations, it is useful to examine Buchberger's original algorithm which can be found in **Figure 2.1**.

From our modern perspective, Buchberger's Algorithm is essentially a *critical pair completion* algorithm. Again, for the sake of discussion, let us imagine we have fixed a monomial order $\prec$. Given an input basis F, Buchberger's Algorithm converts F into a Gröbner basis G for the same ideal $\mathcal{I}(F) = \mathcal{I}(G)$ by adding additional polynomials into the basis which "patch" obstructions of the confluence of the reduction relation $\mapsto_F$. S-polynomials are precisely these "patches," also known as "critical pairs." Let us gain intuition about S-polynomials with the following simple example.

Example 2.3.13 (S-polynomial intuition)**.** *Let*

$$p_1 = \underline{x_1x_2} - x_4 \quad \textit{and} \quad p_2 = \underline{x_2x_3} - x_5.$$

Then, $\mapsto_{\{p_1,p_2\}}$ *is not confluent. To see this, consider the following reductions of* $x_1x_2x_3$*:*

$$x_1x_2x_3 \mapsto_{p_1} x_3x_4 \quad \textit{and} \quad x_1x_2x_3 \mapsto_{p_2} x_1x_5.$$

The obstruction of confluence is the fact that under $\mapsto_F$*,* x_3x_4 *and* x_1x_5 *do not have a common reduct. The S-polynomial* $spol(p_1,p_2)$ *will "patch" this obstruction. Let us recall its definition:*

$$\begin{aligned} spol(p_1,p_2) &= (\frac{\mathsf{lcm}(x_1x_2,x_2x_3)}{x_1x_2})x_4 - (\frac{\mathsf{lcm}(x_1x_2,x_2x_3)}{x_2x_3})x_5 \\ &= (\frac{x_1x_2x_3}{x_1x_2})x_4 - (\frac{x_1x_2x_3}{x_2x_3})x_5 \\ &= x_3x_4 - x_1x_5. \end{aligned}$$

By totality of $\prec$*, either* $x_3x_4 \prec x_1x_5$ *or* $x_1x_5 \prec x_3x_4$*. Let's assume* $x_1x_5 \prec x_3x_4$*, and let* $G = \{p_1,p_2,spol(p_1,p_2)\}$*. Then, the S-polynomial* $spol(p_1,p_2) = \underline{x_3x_4} - x_1x_5$ *contributes the reduction relation* $\mapsto_{spol(p_1,p_2)}$ *to* $\mapsto_G$*, where* $\mapsto_{spol(p_1,p_2)}$ *is defined as*

$$q_1 - c_1m_1x_3x_4 \;\; \mapsto_{spol(p_1,p_2)} \;\; q_1 - c_1m_1x_1x_5$$

for arbitrary monomials c_1m_1 *and polynomials* q_1*. In particular, we have then "patched" the obstruction of confluence we noted above by giving* x_3x_4 *the reduct* x_1x_5*.*

Of course, the above example is trivial in the sense that the obstruction to confluence was "patched" immediately by the computed S-polynomial, by inducing a reduction from x_3x_4 directly to x_1x_5. In general, this need not be the case. The important point, which allows Gröbner bases to be computed through the introduction of S-polynomials, is that each S-polynomial will in a precise sense draw the reduction relation *closer* to being confluent. Let us solidify these intuitions by sketching a proof of the correctness of Buchberger's Algorithm.

In the course of proving the correctness of Buchberger's algorithm, we will first fix some additional notation and then state a number of preliminary results without proof. Proofs of these theorems may be found in **Chapters 6-7** of [CLO07].

Definition 2.3.14 (Head monomial)**.** If $p = \underline{c_1m_1} + q$, then $HM(p) = c_1m_1$.

Definition 2.3.15 (Set of head monomials)**.** Let $G = \{p_1,\ldots,p_k\}$. Then,

$$HM(G) = \{HM(p_1),\ldots,HM(p_k)\}.$$

Definition 2.3.16 (Head monomial ideal)**.** Let $G = \{p_1, \ldots, p_k\}$. Then,

$$\mathcal{I}(HM(G)) = \mathcal{I}(\{HM(p_1), \ldots, HM(p_k)\}).$$

Recall that a ring is Noetherian (as is our ring of polynomials $\mathbb{Q}[\vec{x}]$) if and only if it satisfies the Ascending Chain Condition. This condition will play a key role in the termination of Buchberger's Algorithm, and thus we state it here for convenience.

Theorem 2.3.17 (Ascending Chain Condition)**.** *Let*

$$\mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \mathcal{I}_3 \subseteq \cdots$$

be an ascending chain of ideals in $\mathbb{Q}[\vec{x}]$*. Then, there exists an* $n \in \mathbb{N}^+$ *s.t.*

$$\mathcal{I}_n = \mathcal{I}_{n+1} = \mathcal{I}_{n+2} = \cdots.$$

The following theorem is an important result along the way to the correctness of Buchberger's Algorithm.

Theorem 2.3.18 (Buchberger's S-polynomial Criterion)**.** *Let* $\mathcal{I}$ *be a polynomial ideal. Then* $G = \{p_1, \ldots, p_k\}$ *is a Gröbner basis for* $\mathcal{I}$ *iff*

$$\mathcal{I}(G) = \mathcal{I} \quad \textit{and} \quad \forall 1 \leq i < j \leq k \left(spol(p_i, p_j) \xrightarrow{G} 0 \right).$$

Let us now examine the correctness of Buchberger's Algorithm.

Theorem 2.3.19 (Correctness of Buchberger's Algorithm)**.** *Given a set of monic polynomials* $F = \{p_1, \ldots, p_k\}$ *and a monomial order* $\prec$*, a run of Buchberger's Algorithm upon* F *and* $\prec$ *is guaranteed to terminate and produce a Gröbner basis for* $\mathcal{I}(F)$*.*

Proof. Let us first observe that in each iteration of the loop, the set G generates the same ideal as the input set F. This is certainly true before the loop begins, as G is initialised to F. Then, whenever G is enlarged, it is enlarged by inserting the residue q of an S-polynomial $spol(p_i, p_j)$ with $p_i, p_j \in G$. By **Observation 2.3.5**, $p_i, p_j \in \mathcal{I}(G) \implies spol(p_i, p_j) \in \mathcal{I}(G)$. Then, by **Observation 2.3.10**, it follows that the residue q is contained in $\mathcal{I}(G)$ as well. Thus, the loop maintains the invariant $\mathcal{I}(F) = \mathcal{I}(G)$. Observe that the algorithm only terminates when every S-polynomial between non-equal members of G has residue 0. By **Theorem 2.3.18**, it then follows that if the algorithm terminates, G is indeed a Gröbner basis for $\mathcal{I}(F)$. So, it will suffice to prove termination. Consider a pass of the loop. If the termination condition is not met (that is, $G' \neq G$), then there must exist some collection of m

pairs $P = \{\langle p_{i_1}, p_{j_1}\rangle, \ldots, \langle p_{i_m}, p_{j_m}\rangle\} \subset (G' \times G')$ s.t. the S-polynomial of each pair $spol(p_{i_a}, p_{j_a})$ gives rise to a non-zero residue q_a w.r.t. G' (i.e., $spol(p_{i_a}, p_{j_a}) \xrightarrow{G'} q_a \neq 0$) so that $G = G' \cup \{q_1, \ldots, q_m\}$. So, $G' \subset G$ and thus $\mathcal{I}(HM(G')) \subseteq \mathcal{I}(HM(G))$. In fact, since $G \neq G'$, a stronger property — that $\mathcal{I}(HM(G))$ is strictly larger than $\mathcal{I}(HM(G'))$ — will hold: $\mathcal{I}(HM(G')) \subset \mathcal{I}(HM(G))$. Let us see why this is so. Consider q_a which has been adjoined to G' in the process of forming G. The fact that q_a could not be further reduced by the reduction relation $\mapsto_{G'}$ induced by G' means that $HM(q_a)$ is not the head monomial of any polynomial in G'. But, since $q_a \in G$, it then follows that $HM(q_a)$ is the head monomial of a polynomial in G. Furthermore, since $HM(q_a)$ is non-zero, we know that no monomial in $HM(G')$ divides $HM(q_a)$. As a monomial m is a member of a monomial ideal $\mathcal{I}(\{m_1, \ldots, m_k\})$ iff m is divisible by some m_i, it then follows that no monomial in $\mathcal{I}(HM(G'))$ divides $HM(q_a)$. Thus, $HM(q_a) \notin \mathcal{I}(HM(G'))$. And so, $\mathcal{I}(HM(G')) \subset \mathcal{I}(HM(G))$. So, now we have that if the loop does not terminate, then G has been enlarged so that the ideals $\mathcal{I}(HM(G))$ from successive iterations of the loop form an ascending chain of ideals in $\mathbb{Q}[\vec{x}]$. But, by **Theorem 2.3.17**, this ascending chain of ideals will stabilise after finitely many iterations. Thus, after finitely many iterations, $\mathcal{I}(HM(G)) = \mathcal{I}(HM(G'))$ will hold. But then there could be no non-zero residues of S-polynomials between members of G', and thus $G = G'$ and termination is proved. □

It will also be useful later to have the notion of a *reduced* Gröbner basis. Recall again our convention that all Gröbner bases consist of monic polynomials.

Definition 2.3.20 (Reduced Gröbner basis)**.** A Gröbner basis G is *reduced* iff

$$\forall p \in G \quad \text{no monomial of } p \text{ lies in } \mathcal{I}(HM(G \setminus \{p\})).$$

2.3.4 Superfluous S-polynomial Criteria

After understanding why Buchberger's Algorithm is correct as we have above, it is natural to begin considering ways in which its processing may be made more efficient. The algorithm as presented is extremely naive, and in practical implementations, one wishes to avoid as much unnecessary processing as possible.

One of the first and most important sources of efficiency in Gröbner basis construction algorithms has been the recognition of so-called "superfluous S-polynomial criteria." The impetus for these criteria are the following two points:

- In an execution of Buchberger's Algorithm, if an S-polynomial $spol(p_i, p_j)$ reduces to zero w.r.t. the current basis G', then $spol(p_i, p_j)$ does not contribute a residue to the Gröbner basis G being constructed.

- When computing Gröbner bases in practice, huge computational resources are often expended upon precisely these "reductions to zero."

Thus, in the context of Buchberger's Algorithm, it is advantageous to develop computationally efficient sufficient conditions for recognising when a given S-polynomial would in fact reduce to zero w.r.t. the current basis being constructed. In these cases, one can then avoid performing any reductions of such superfluous S-polynomials, as they will not contribute any residues to the Gröbner basis.

The first such criteria recognised were put forth by Buchberger in his paper [Buc79]. For instance, the criterion we will refer to as Buchberger-1 in this dissertation is the following:

Criterion 1. *If* $p_1 = \underline{m_1} + q_1$ *and* $p_2 = \underline{m_2} + q_2$ *and* $\mathsf{lcm}(m_1, m_2) = m_1 m_2$*, then* $\mathsf{spol}(p_1, p_2)$ *is superfluous.*

Since Buchberger's original paper on superfluous S-polynomial criteria, a number of others have been developed, including those of Gebauer and Möller ([GM88]) and those which are implicit in the linear algebra techniques underlying Faugère's algorithms F4 [Fau99] and F5 [Fau02].

A key contribution of our dissertation consists of an exploration of novel algorithms — that is, algorithms which exhibit behavior very different than Buchberger's Algorithm or F4 or F5 — for computing Gröbner bases. These novel algorithms have been motivated by efficiency considerations for certain classes of practical problems encountered during program verification (cf. **Chapter 5**).

A principle difficulty in exploiting superfluous S-polynomial criteria in the context of these non-standard approaches to computing Gröbner bases, however, is that these criteria are usually proved correct w.r.t. a fixed Gröbner basis construction algorithm. For instance, Buchberger-1 above was in [Buc79] proved correct w.r.t. Buchberger's Algorithm. For more involved criteria such as Buchberger-2 as analysed in the next chapter, proving their correctness ("admissibility") in the context of non-standard Gröbner basis construction algorithms is a serious challenge.

In **Chapter 3**, we will develop an abstract theory of Gröbner basis construction algorithms in which many different approaches to computing Gröbner basis may be

analysed in a uniform setting. A key aspect of this work will be developing techniques with which the admissibility of these superfluous S-polynomial criteria can be analysed and proven abstractly so that they can soundly be incorporated into a multitude of different Gröbner basis construction algorithms.

The practical fruits of this theoretical work will be examined in **Chapter 5**, where two particularly interesting approaches to computing Gröbner bases are presented, proved correct, implemented and experimentally evaluated.

2.3.5 Complex Satisfiability and Weak Nullstellensatz

As our interest in Gröbner bases is chiefy motivated by a desire for improved algorithms for deciding the satisfiability of systems of polynomial equations and disequations over the complex numbers, it is useful to examine how Gröbner bases can contribute to this goal.

In particular, it will turn out that an algorithmic test for *ideal membership* is sufficient for deciding complex satisfiability. As we have seen, given an ideal $\mathcal{I}(\{p_1,\ldots,p_k\})$ and a polynomial q, we can decide whether or not $q \in \mathcal{I}(\{p_1,\ldots,p_k\})$ simply by forming a Gröbner basis G for $\mathcal{I}(\{p_1,\ldots,p_k\})$ and checking if $q \xrightarrow{G} 0$. The result which connects ideal membership to complex satisfiability is known as Hilbert's Weak Nullstellensatz [CLO07] (stated below over $\mathbb{C}$ for concreteness).

Theorem 2.3.21 (Hilbert's Weak Nullstellensatz (over $\mathbb{C}$))**.** *Let* $\mathcal{I} = \mathcal{I}(\{p_1,\ldots,p_k\}) \subseteq \mathbb{Q}[\vec{x}]$. *Then,*

$$\mathcal{V}_{\mathbb{C}}(\mathcal{I}) = \emptyset \quad \Longleftrightarrow \quad \mathcal{I} = \mathbb{Q}[\vec{x}].$$

That is, the system of polynomial equations

$$\begin{aligned} p_1(\vec{x}) &= 0, \\ &\vdots \\ p_k(\vec{x}) &= 0 \end{aligned}$$

has a solution in $\mathbb{C}^n$ iff $\mathcal{I}(\{p_1,\ldots,p_k\}) \neq \mathbb{Q}[\vec{x}]$.

From our perspective, it is more telling to restate this result in a form that references ideal membership explicitly. This is done using the following observation.

Observation 2.3.22. Given a polynomial ideal $\mathcal{I}$ over $\mathbb{Q}[\vec{x}]$,

$$\mathcal{I} = \mathbb{Q}[\vec{x}] \quad \Longleftrightarrow \quad 1 \in \mathcal{I}.$$

Thus, we obtain the following restatement of Hilbert's Weak Nullstellensatz connecting complex satisfiability and ideal membership:

Theorem 2.3.23 (Hilbert's Weak Nullstellensatz - revised)**.** *Let* $\mathcal{I} = \mathcal{I}(\{p_1, \ldots, p_k\}) \subseteq \mathbb{Q}[\vec{x}]$. *Then,*

$$\left(\langle \mathbb{C}, +, -, *, 0, 1 \rangle \models \neg \exists \vec{x} (\bigwedge_{i=1}^{k} p_i = 0) \right) \quad \Longleftrightarrow \quad 1 \in \mathcal{I}.$$

So, we have now reduced the problem of satisfiability for a system of polynomial *equations* to ideal membership. We will need one last simple ingredient so as to extend our machinery to handle systems of both polynomial equations and *disequations*.

Observation 2.3.24.

$$\langle \mathbb{C}, +, -, *, 0, 1 \rangle \models \forall x \, (x \neq 0 \quad \Longleftrightarrow \quad \exists y (x * y + 1 = 0)) \, .$$

Thus, a polynomial disequation in a system of polynomial equations and disequations may be converted to an equivalent equality by introducing a slack variable and replacing the disequation as above.

We now see explicitly how to decide the satisfiability of a system of polynomial equations and disequations over the complex numbers using a combination of Gröbner bases and Hilbert's Weak Nullstellensatz.

Theorem 2.3.25 (Gröbner solution to complex satisfiability)**.** *Let S be the system of polynomial equations and disequations*

$$\begin{aligned} p_1(\vec{x}) &= 0, \\ &\vdots \\ p_a(\vec{x}) &= 0, \\ p_{a+1}(\vec{x}) &\neq 0, \\ &\vdots \\ p_k(\vec{x}) &\neq 0. \end{aligned}$$

Then, the satisfiability of S over the complex numbers can be decided by the following algorithm.

1. *Convert S into an equivalent system of polynomial equations* S' *using* ***Observation 2.3.24****. The polynomials in* S' *will now be in the ring* $\mathbb{Q}[\vec{x}, y_1, \ldots, y_{k-a}]$. *Let P be the set of polynomials in* S'.

2. *Fix a monomial order $\prec$ and compute a Gröbner basis G for $\mathcal{I}(P)$ w.r.t. $\prec$.*

3. *By **Theorem 2.3.23**, it will suffice to check whether or not $1 \in \mathcal{I}(P)$ by computing the residue of 1 w.r.t. G. Then, S is unsatisfiable over the complex numbers iff $1 \xrightarrow{G} 0$.*

□

Chapter 3

Abstract Gröbner Bases and Superfluous S-polynomials

3.1 Introduction

In this chapter, we present an abstract theory of Gröbner basis procedures and use it to analyse superfluous S-polynomial criteria in a strategy-independent manner. Under this theory, different algorithms for computing Gröbner bases will formally correspond to *different strategies* for orchestrating a small set of inference rules. The technique of *proof orders* will be used to derive a generalisation of S-polynomial superfluousness in terms of transfinite induction along an ordinal parameterised by a monomial order. This generalisation expresses S-polynomial superfluousness in a new, more abstract way which is independent of the Gröbner basis construction algorithm used: It states that an S-polynomial is superfluous if, in its absence, a certain class of formal proofs can still be transformed into *smaller* "equivalent" proofs with respect to a well-founded ordering upon these proofs. This statement is made precise with **Observation 3.4.1**. We will then use this generalisation to prove that three superfluous S-polynomial criteria are admissible with respect to any Gröbner basis construction algorithm corresponding to a correct strategy in our system. These superfluous S-polynomial criteria are important for efficient Gröbner basis construction and will be further exploited in **Chapter 5** as the basis of term-indexing techniques for a new class of Gröbner basis construction algorithms targeted to the needs of SMT solvers.

3.1.1 Related Work

There is a rich history of work on connections between Gröbner bases, critical-pair completion, and automated theorem proving. Already in 1984 [Buc84], the view of Gröbner basis construction as critical-pair completion was recognised by Buchberger and used to fruitfully extend Gröbner basis methods to new domains. Following this, Buchberger made connections between Gröbner bases, completion and resolution theorem proving explicit in 1987 [Buc87].

The work most relevant to this chapter, however, concerns *abstract frameworks* for analysing both completion and Gröbner basis procedures. In the case of completion, for instance, such a framework allows one to view different completion algorithms as being particular *strategies* for sequencing a small set of inference rules. In doing so, one is able use uniform methods for proving results about a multitude of different completion procedures simultaneously. There are a number of frameworks upon which we build.

The first is the Bachmair-Dershowitz theory of *Abstract Completion* [BD94]. The second is the Bachmair-Ganzinger framework developed for presenting Buchberger's algorithm as a *constraint-based* completion procedure [BG94]. In the end, we found it necessary to derive our own framework, based upon [BD94], for analysing the Gröbner basis algorithms presented in this thesis. We call this framework *Abstract Gröbner Bases*. It is essentially [BD94] with its term machinery instantiated upon $\mathbb{Q}[\vec{x}]$ and restricted to ground equations. Equivalently, it can be seen as a simplification of [BG94] in which the hierarchical constraint system used explicitly for coefficient normalisation is eliminated in lieu of standard polynomial representation machinery common in modern Gröbner basis theory.

Once the framework of Abstract Gröbner Bases is presented, we will use it to investigate the admissibility of superfluous S-polynomial criteria in a strategy-independent manner. These critera, as explained in **Chapter 5**, are crucial to term indexing techniques used in new Gröbner basis algorithms we present which are based upon saturation and simplification loops used in high-performance superposition theorem proving. In principle, the results we prove about superfluous S-polynomial criteria could be suitably translated and the proofs carried out using the other two frameworks mentioned above, though the resulting arguments would surely be much more technically unmanageable than those presented here.

Orient $$\frac{S \cup \{\underline{cm} + q\}, G}{S, G \cup \{\underline{m} + (\frac{1}{c})q\}}$$

Superpose $$\frac{S, G \cup \{p_1, p_2\}}{S \cup \{\mathsf{spol}(p_1, p_2)\}, G \cup \{p_1, p_2\}}$$

Delete $$\frac{S \cup \{0\},\ G}{S, G}$$

Simplify-S $$\frac{S \cup \{c_1 m_1 m_2 + q_1\}, G \cup \{\underline{m_2} + q_2\}}{S \cup \{q_1 - c_1 m_1 q_2\}, G \cup \{\underline{m_2} + q_2\}}$$

Simplify-H $$\frac{S, G \cup \{\underline{m_1 m_2} + q_1, \underline{m_2} + q_2\}}{S \cup \{q_1 - m_1 q_2\}, G \cup \{\underline{m_2} + q_2\}} \textbf{ if } m_1 \neq 1$$

Simplify-T $$\frac{S, G \cup \{\underline{m} + c_1 m_1 m_2 + q_1, \underline{m_2} + q_2\}}{S, G \cup \{\underline{m} - c_1 m_1 q_2 + q_1, \underline{m_2} + q_2\}}$$

Figure 3.1: Abstract GB Inference Rules

3.1.2 Our Contribution

The results of this chapter were obtained jointly with Dr. Leonardo de Moura, with both of us contributing equally. The results of this chapter were published as [PdM09a].

3.2 Theory of Abstract Gröbner Bases

We freely utilise notation and concepts introduced in **Section 2.3**. Let us now present the framework of Abstract Gröbner Bases. We express the system as a collection of inference rules.

The inference rules (cf. **Figure 3.1**) work on pairs of sets of polynomials (S, G). In all rules, the coefficients c and c_1 are assumed to be non-zero. We use $(S_1, G_1) \vdash (S_2, G_2)$ to indicate that (S_1, G_1) can be transformed to (S_2, G_2) by applying one of the inference rules in **Figure 3.1**.

Theorem 3.2.1. $(S_1, G_1) \vdash (S_2, G_2) \implies \mathcal{I}(S_1 \cup G_1)) = \mathcal{I}(S_2 \cup G_2))$.

Proof. Easy by observing (i) every rule that extends (S_1, G_1) does so by adding polynomials already in $\mathcal{I}(S_1 \cup G_1)$, (ii) reducing a polynomial p using q when p and q

are in (S_1, G_1) does not change $\mathcal{I}(S_1 \cup G_1)$, and (iii) a polynomial p is removed from $(S_1 \cup G_1)$ only when $p = 0$. □

Definition 3.2.2 (Procedure)**.** *A Gröbner basis procedure* $\mathfrak{G}$ *is a program that accepts a set of polynomials* $\{p_1, \ldots, p_k\}$*, a monomial order* $\prec$*, and uses the rules in* ***Figure 3.1*** *to generate a (finite or infinite) sequence* $(S_1 = \{p_1, \ldots, p_k\},\ G_1 = \emptyset) \vdash (S_2, G_2) \vdash (S_3, G_3) \vdash \ldots$*. This sequence is called a* run *of* $\mathfrak{G}$*.*

Given a set of monic polynomials G, the set of S-polynomials $\mathsf{SP}(G)$ is defined as the set

$$\{\mathsf{spol}(p_1, p_2) \mid p_1, p_2 \in G\}.$$

Definition 3.2.3 (Correct Procedure)**.** *A Gröbner basis procedure* $\mathfrak{G}$ *is said to be correct iff it produces only finite runs* $(S_1,\ G_1 = \emptyset) \vdash \ldots \vdash (S_n = \emptyset,\ G_n)$*, and*

$$\mathsf{SP}(G_n) \subseteq (S_1 \cup S_2 \cup \ldots \cup S_{n-1})\,.$$

Theorem 3.2.4. *Let* $\mathfrak{G}$ *be a correct Gröbner basis procedure, then for any run* $(S_1,\ G_1 = \emptyset) \vdash \ldots \vdash (S_n = \emptyset,\ G_n)$*,* G_n *is a Gröbner basis for* $\mathcal{I}(S_1)$*.*

The proof of **Theorem 3.2.4**, which follows from **Theorem 3.3.5** below, uses a technique called *proof orders*. We will study this in detail in the next section.

Definition 3.2.5 (Eager S-simplification)**.** *Given a Gröbner basis procedure* $\mathfrak{G}$*, we say* $\mathfrak{G}$ *implements eager S-simplification iff* $\mathfrak{G}$ *only applies* Orient *to* $p \in S_i$ *when* Simplify-S *cannot be applied to* p*.*

Observation 3.2.6. Given a Gröbner basis procedure $\mathfrak{G}$ using eager S-simplification, then for any run $(S_1, G_1) \vdash (S_2, G_2) \vdash \ldots$, for all $j \geq 1$, there is no $\underline{m_1} + q_1$ and $\underline{m_2} + q_2$ in G_j such that $m_1 = m_2$ and $q_1 \neq q_2$. Moreover, in this case, the condition $m_1 \neq 1$ in the rule Simplify-H is only restricting self simplifications.

Definition 3.2.7 (Fairness)**.** *A Gröbner basis procedure* $\mathfrak{G}$ *is said to be fair iff for any run* $(S_1, G_1) \vdash (S_2, G_2) \vdash \ldots$

$$\mathsf{SP}(\bigcup_{i\geq 1} \bigcap_{j \geq i} G_j) \subseteq \bigcup_{i \geq 1} S_i.$$

Theorem 3.2.8. *If a Gröbner basis procedure* $\mathfrak{G}$ *implements eager S-simplification, is fair, and* Superpose *is applied at most once for any pair of polynomials in* $\bigcup_{i\geq 1} G_i$*, then* $\mathfrak{G}$ *is correct.*

Proof. We just need to show that every run of $\mathfrak{G}$ is finite. This follows from Dickson's lemma, and the fact that any infinite run will contain an infinite number of Superpose steps. □

Example 3.2.9. *Let F be the set of polynomials:*

$$\{x^2y-1,\ xy^2-y\}.$$

*Then, using the inference rules in **Figure 3.1**, we can generate the run in **Figure 3.2**. A reduced Gröbner basis for F is contained in the final state $(\emptyset,\ \{y-1,\ x-1\})$.*

As an exercise in gaining familiarity with the inference rules, we illustrate how they can be used to simulate Buchberger's algorithm in **Figure 3.3**.

3.3 Proof Orders

In this section, we use the technique of proof orders prove **Theorem 3.2.4** and some important related results. By paying close attention to how S-polynomials are actually used in the proof of a key lemma, **Lemma 3.3.3**, we will be able to derive a "strategy-independent" generalisation of S-polynomial superfluousness. This generalisation will be made precise and applied to a number of superfluous S-polynomial criteria in the next section.

In the following, we assume that

$$(F = S_1, G_1 = \emptyset) \vdash \ldots \vdash (S_n = \emptyset, G_n)$$

is an arbitrary run of a correct Gröbner basis procedure $\mathfrak{G}$. We use S_* to denote the set $S_1 \cup \ldots \cup S_n$ and G_* to denote the set $G_1 \cup \ldots \cup G_n$.

An *equational step* in (S_*, G_*) is a tuple $\langle s, p, cm, t\rangle$, where s, p and t are polynomials, cm is a monomial, $p \in S_* \cup G_*$, and $t = s - cmp$. We use

$$s \xleftarrow{\langle p, cm\rangle} t$$

to denote the equational step $\langle s, p, cm, t\rangle$.

Observation 3.3.1. Let $\langle s, p, cm, t\rangle$ be an equational step, then for any monomial $c'm'$ in p, s or t contains the power-product $m'm$.

A *right rewrite step* in (S_*, G_*) is a tuple $\langle s, p, m, t\rangle$, where s, p and t are polynomials, m is a monic monomial, and $p \in G_*$ s.t. if s is of the form $c_s m m_p + q_s$ and p is of

$\{x^2y-1,\ xy^2-y\},\ \emptyset$

$\vdash$ Orient: x^2y-1

$\{xy^2-y\},\ \{\underline{x^2y}-1\}$

$\vdash$ Orient: xy^2-y

$\emptyset,\ \{\underline{x^2y}-1,\ \underline{xy^2}-y\}$

$\vdash$ Superpose: spol$(\underline{x^2y}-1,\ \underline{xy^2}-y)=xy-y$

$\{xy-y\},\ \{\underline{x^2y}-1,\ \underline{xy^2}-y\}$

$\vdash$ Orient: $xy-y$

$\emptyset,\ \{\underline{x^2y}-1,\ \underline{xy^2}-y,\ \underline{xy}-y\}$

$\vdash$ Simplify-H: $\underline{xy}-y$ over $\underline{x^2y}-1$

$\{xy-1\},\ \{\underline{xy^2}-y,\ \underline{xy}-y\}$

$\vdash$ Simplify-S: $\underline{xy}-y$ over $xy-1$

$\{y-1\},\ \{\underline{xy^2}-y,\ \underline{xy}-y\}$

$\vdash$ Orient: $y-1$

$\emptyset,\ \{\underline{xy^2}-y,\ \underline{xy}-y,\ \underline{y}-1\}$

$\vdash$ Simplify-H: $\underline{y}-1$ over $\underline{xy^2}-y$

$\{xy-y\},\ \{\underline{xy}-y,\ \underline{y}-1\}$

$\vdash$ Simplify-S: $\underline{xy}-y$ over $xy-y$

$\{0\},\ \{\underline{xy}-y,\ \underline{y}-1\}$

$\vdash$ Delete

$\emptyset,\ \{\underline{xy}-y,\ \underline{y}-1\}$

$\vdash$ Simplify-H: $\underline{y}-1$ over $\underline{xy}-y$

$\{x-y\},\ \{\underline{y}-1\}$

$\vdash$ Simplify-S: $\underline{y}-1$ over $x-y$

$\{x-1\},\ \{\underline{y}-1\}$

$\vdash$ Orient: $x-1$

$\emptyset,\ \{\underline{y}-1,\ \underline{x}-1\}$

$\vdash$ Superpose: spol$(\underline{y}-1,\underline{x}-1)=x-y$

$\{x-y\},\ \{\underline{y}-1,\ \underline{x}-1\}$

$\vdash$ Simplify-S: $\underline{y}-1$ over $x-y$

$\{x-1\},\ \{\underline{y}-1,\ \underline{x}-1\}$

$\vdash$ Simplify-S: $\underline{x}-1$ over $x-1$

$\{0\},\ \{\underline{y}-1,\ \underline{x}-1\}$

$\vdash$ Delete:

$\emptyset,\ \{\underline{y}-1,\ \underline{x}-1\}$

Figure 3.2: A *run* for $\{x^2y-1,\ xy^2-y\}$ w.r.t. DegLex with $x \prec y$

Input: $\langle S = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\vec{x}], \prec \rangle$
Output: G s.t. G is a GBasis of S w.r.t. $\prec$
Apply Orient to every member of S
Apply Superpose between every $p, p' \in G$ $(p \neq p')$
while $S \neq \emptyset$ **do**
  Choose spol$(p, p') \in S$
  Apply Simplify-S to spol$(p, p') \in S$ as long as possible
  Call the resulting simplified polynomial (in S) q
  if $q \neq 0$ **then**
    Apply Orient to q
    Apply Superpose between every $p, p' \in G$ $(p \neq p')$
      for which Superpose has not been previously
      applied
  else
    Apply Delete to q
  end if
end while

Figure 3.3: Rule-based Simulation of Buchberger's Algorithm

the form $\underline{m_p} + q_p$, then $t = s - c_s mp = q_s - c_s mq_p$. Intuitively, p is a polynomial being used as a rewrite rule, and m specifies that the monomial $c_s mm_p$ of s will be "rewritten" to $-c_s mq_p$. We use

$$s \xrightarrow{\langle p,m\rangle} t$$

to denote the right rewrite step $\langle s,p,m,t\rangle$.

Similarly, a *left rewrite step* in (S_*,G_*) is a tuple $\langle t,p,m,s\rangle$, where s, p, t and m are defined as in the right rewrite step case. We use

$$t \xleftarrow{\langle p,m\rangle} s$$

to denote the left rewrite step $\langle t,p,m,s\rangle$. A *rewrite step* is a left or right rewrite step. For every rewrite step, we say s is the *source* and t is the *target*. Note that $t \prec s$.

A *proof step* is an equational step or a rewrite step. We use $s \simeq_F t$ to denote that $s \in \mathcal{I}(F)$ iff $t \in \mathcal{I}(F)$. Observe that $\mathcal{I}(F) = \mathcal{I}(S_* \cup G_*)$, hence for all proof steps $p \in \mathcal{I}(F)$, and $s \simeq_F t$.

A *proof* Pr for $p \simeq_F q$ in (S_*,G_*) is a sequence of proof steps

$$\langle s_1,p_1,c_1m_1,t_1\rangle \ldots \langle s_k,p_k,c_km_k,t_k\rangle$$

such that, $s_1 = p$, $t_k = q$, $t_i = s_{i+1}$ for $i \in \{1,\ldots,k-1\}$. We use $lhs(\mathrm{Pr})$ to denote s_1 and $rhs(\mathrm{Pr})$ to denote t_k.

For example, let F be the set $\{xy-y, x^2y-1\}$. Hence, for any run, $xy-y \in S_0$. Now, assume $\underline{x^2y} - 1 \in G_*$. Then,

$$y \xleftarrow{\langle xy-y,-x\rangle} y + x^2y - xy \xleftarrow{\langle xy-y,-1\rangle} x^2y \xrightarrow{\langle x^2y-1,1\rangle} 1$$

is a proof for $y \simeq_F 1$.

A *rewrite proof* Pr is a proof containing k rewrite steps such that p_i is in G_n for $i \in \{1,\ldots,k\}$, and there is a $j \in \{0,\ldots,k\}$, where the first j steps are right rewrite steps, and the others are left rewrite steps.

For example, assume G_n contains the polynomials $\{\underline{x}+1,\ \underline{y}+z,\ \underline{w^2}-1\}$. Then, the following proof is a rewriting proof for $xy+2 \simeq_F w^2z+2$.

$$xy+2 \xrightarrow{\langle \underline{x}+1,y\rangle} -y+2 \xrightarrow{\langle \underline{y}+z,1\rangle} z+2 \xleftarrow{\langle \underline{w^2}-1,z\rangle} w^2z+2$$

We say two proofs Pr_1 and Pr_2 in (S_*,G_*) are *equivalent* if $lhs(\mathrm{Pr}_1) = lhs(\mathrm{Pr}_2)$ and $rhs(\mathrm{Pr}_1) = rhs(\mathrm{Pr}_2)$.

The *cost* of a proof step is a pair where the first component is a multi-set of polynomials and the other a polynomial, and is defined as:

1. For $s \xleftarrow{\langle p,cm\rangle} t$, the cost is $(\{s,t\},0)$.
2. For $s \xrightarrow{\langle p,m\rangle} t$ and $t \xleftarrow{\langle p,m\rangle} s$, the cost is $(\{s\},p)$.

Two different cost pairs are compared using the lexicographic product order $\ll$ of $(\prec_M,\prec)$, where $\prec_M$ is the multi-set extension of the order $\prec$ on polynomials. Proof steps are compared by comparing their costs. The overall cost of a proof Pr is the multi-set of the costs of all its proof steps, and two different multi-sets of costs are compared using the multi-set extension $\ll_M$ of $\ll$. Finally, proofs are compared by comparing their costs, and we use $\mathrm{Pr}' \sqsubset \mathrm{Pr}$ to denote that proof Pr' is smaller than proof Pr.

Lemma 3.3.2. *The order $\sqsubset$ is well-founded.*

Proof. This is an immediate consequence of the following facts: the order $\prec$ is well-founded, the multi-set extension of a well-founded order is well-founded, and the lexicographic product order of well-founded orders is well-founded. □

Lemma 3.3.3. *Let* Pr *be a proof in (S_*,G_*) that is not a rewrite proof. Then, there exists a proof* Pr' *in (S_*,G_*) such that* Pr' *is equivalent to* Pr *and* $\mathrm{Pr}' \sqsubset \mathrm{Pr}$.

Proof. If Pr is not a rewrite proof, then there are three possible reasons:

1. Pr contains an equational step.
2. Pr contains a rewrite step $\langle s_i,p_i,m_i,s_{i+1}\rangle$, and p_i is not in G_n.
3. Pr contains a *peak* of the form

$$t_1 \xleftarrow{\langle p_1,m_1\rangle} s \xrightarrow{\langle p_2,m_2\rangle} t_2$$

for p_1 and p_2 in G_n.

In the following, we consider each of these three cases separately.

1. Assume Pr contains an equational step

$$s \xleftarrow{\langle p,cm\rangle} t$$

By definition of equational step, $t = s-(cm)p$. First, assume $p \in S_*$, then since $S_n = \emptyset$, p is removed from some $S_{j<n}$ using Orient, Delete or Simplify-S. The case where $p \in G_*$ is similar to the case where p is removed from some $S_{j<n}$ using Orient.

(a) Assume Orient was used to remove p. Let p be of the form $\underline{c_p m_p} + q_p$, then $p' = (\frac{1}{c_p})p$ is in G_{j+1}. By **Observation 3.3.1**, s or t must contain the power-product $m_p m$. First, let us assume that s contains $c_s m_p m$ and t does not. Then, $c_s = c_p c$ because t does not contain the power-product $m_p m$, and by simple algebraic manipulation:

$$\begin{aligned} t &= s-(cm)p = s-(\frac{c_s}{c_p}m)p = s-(c_s m)((\frac{1}{c_p})p) \\ &= s-(c_s m)p'. \end{aligned}$$

Let Pr' be the proof that is obtained by replacing the equational step with:

$$s \xrightarrow{\langle p',m\rangle} t$$

Similarly, if t contains the power-product $m_p m$ and s does not, we replace the the equational step with the rewrite step:

$$s \xleftarrow{\langle p',m\rangle} t$$

Finally, if both of them contain the power-product $m_p m$, let c_t be the coefficient of $m_p m$ in t. Then, by the definition of equational step, $c_t = c_s - c_p c$. Let s' be the polynomial $s-(c_s m)p'$. By algebraic manipulation, we have:

$$\begin{aligned} s' &= s-(c_s m)p' = s-((c_p c + c_t)m)p' \\ &= s-(cm)(c_p p') - (c_t m)p' \\ &= s-(cm)p - (c_t m)p' \\ &= t-(c_t m)p'. \end{aligned}$$

In this case, let Pr' be the proof that is obtained by replacing the equational step with:

$$s \xrightarrow{\langle p',m\rangle} s' \xleftarrow{\langle p',m\rangle} t$$

In all three cases, the rewrite steps are smaller than the equational step, because $\{s\} \prec_M \{s,t\}$ and $\{t\} \prec_M \{s,t\}$. This shows that the new proof $\mathrm{Pr}' \sqsubset \mathrm{Pr}$.

Before we consider the next case, note that the case where $p \in G_*$ can be handled as above. The only difference is that $p' = p$ when $p \in G_*$.

(b) Assume that Delete was used to remove p, then $p = 0$ and $s = t$, and the equational step can be removed from the proof. Therefore, $\mathrm{Pr}' \sqsubset \mathrm{Pr}$.

(c) Assume p is of the form $c_p m_p m_r + q_p$ and Simplify-S was applied to p using a polynomial $r \in G_j$ of the form $\underline{m_r} + q_r$. Let p' be $-c_p m_p q_r + q_p$, then p' is in S_{j+1}. By **Observation 3.3.1**, s or t must contain the power-product $m_p m_r m$. Let us assume both of them contain $m_p m_r m$, and c_s and c_t are the coefficients of $m_p m_r m$ in s and t respectively. Recall that c_t must be $c_s - c_p c$. Now, let s' be the polynomial $s - (c_s m_p m)r$ and t' be the polynomial $t - (c_t m_p m)r$. Note that $s' \prec s$ and $t' \prec t$. By simple algebraic manipulation we can show that $t' = s' - (cm)p'$. Now, let Pr' be the proof that is obtained by replacing the equational step with:

$$s \xrightarrow{\langle r, m_p m\rangle} s' \xleftarrow{\langle p', cm\rangle} t' \xleftarrow{\langle r, m_p m\rangle} t$$

All three new proof steps are smaller than the original equational step because $\{s\} \prec_M \{s,t\}$, $\{t\} \prec_M \{s,t\}$, and $\{s',t'\} \prec_M \{s,t\}$. This shows that the new proof $\mathrm{Pr}' \sqsubset \mathrm{Pr}$. If s does not contain the power-product $m_p m_r m$, then the first rewrite step is not needed. Similarly, if t does not contain the power-product $m_p m_r m$ the last rewrite step is not needed.

2. Assume Pr contains a rewrite step $\langle s, p, m, t\rangle$, and p is not in G_n. Without loss of generality, assume it is a right rewrite step

$$s \xrightarrow{\langle p, m\rangle} t$$

Since p is not in G_n, it was removed from some $G_{j<n}$ using Simplify-H or Simplify-T and a polynomial $r \in G_j$ of the form $\underline{m_r} + q_r$.

(a) Assume Simplify-H was applied to p using r, and p is of the form $\underline{m_p m_r} + q_p$. Note that $m_p \neq 1$ because of the side condition of Simplify-H, therefore $r \prec p$. Let p' be the polynomial $-m_p q_r + q_p$, then p' is in S_{j+1}. Since $\langle s, p, m, t\rangle$ is a right rewrite step, s must contain the monomial $c_s m_p m_r m$. By the definition of right rewrite rule, $t = s - (c_s m)p$. Now, let s' be the polynomial $s - (c_s m_p m)r$. Thus, by algebraic manipulation, we can show that $t = s' - (c_s m)p'$. Let Pr' be the proof that is obtained by replacing the rewrite step with:

$$s \xrightarrow{\langle r, m_p m\rangle} s' \xleftarrow{\langle p', c_s m\rangle} t$$

The new equational step is smaller than the original step because $s' \prec s$ and $t \prec s$, and consequently $\{s',t\} \prec_M \{s\}$. The cost of the original rewrite step

is $(\{s\}, p)$. The cost of the new rewrite step is $(\{s\}, r)$, and is smaller than $(\{s\}, p)$ because $r \prec p$.

(b) Assume Simplify-T was applied to p using r, and p is of the form $\underline{m'_p} + c_p m_p m_r + q_p$. Let p' be the polynomial $\underline{m'_p} - c_p m_p q_r + q_p$, and thus p' is in G_{j+1}. Then, this case can be handled similarly to case 1c for Simplify-S.

3. Assume Pr contains a peak of the form

$$t_1 \xleftarrow{\langle p_1, m'_1 \rangle} s \xrightarrow{\langle p_2, m'_2 \rangle} t_2$$

for p_1 and p_2 in G_n. Assume p_1 and p_2 are of the form $\underline{m_1} + q_1$ and $\underline{m_2} + q_2$ respectively. Now, we consider two cases: $m'_1 m_1 \neq m'_2 m_2$ and $m'_1 m_1 = m'_2 m_2$.

(a) Assume $m'_1 m_1 \neq m'_2 m_2$, then s must be of the form $q_s + c_1 m'_1 m_1 + c_2 m'_2 m_2$. Moreover, we must have

$$\begin{aligned} t_1 &= q_s - c_1 m'_1 q_1 + c_2 m'_2 m_2, \\ t_2 &= q_s + c_1 m'_1 m_1 - c_2 m'_2 q_2. \end{aligned}$$

Let s' be the polynomial $q_s - c_1 m'_1 q_1 - c_2 m'_2 q_2$. Let Pr$'$ be the proof that is obtained by replacing the peak with:

$$t_1 \xleftarrow{\langle p_2, c_2 m'_2 \rangle} s' \xleftarrow{\langle p_1, c_1 m'_1 \rangle} t_2$$

The polynomials t_1, t_2 and s' are smaller than s, hence $\{t_1, s'\} \prec_M \{s\}$, and $\{s', t_2\} \prec_M \{s\}$. Therefore both equational steps are smaller than the rewrite steps in the peak.

(b) Assume $m'_1 m_1 = m'_2 m_2$, then s must be of the form $q_s + cm\tau_{1,2}$ where $\tau_{1,2} = \mathsf{lcm}(m_1, m_2)$. Then, we must have

$$\begin{aligned} t_1 &= q_s - cm(\frac{\tau_{1,2}}{m_1})q_1 \\ t_2 &= q_s - cm(\frac{\tau_{1,2}}{m_2})q_2 \end{aligned}$$

Moreover, $\mathsf{spol}(p_1, p_2) = \frac{\tau_{1,2}}{m_1} q_1 - \frac{\tau_{1,2}}{m_2} q_2$ must be in S_*. Let Pr$'$ be the proof that is obtained by replacing the peak with:

$$t_1 \xleftarrow{\langle \mathsf{spol}(p_1, p_2), -cm \rangle} t_2$$

Since $\{t_1, t_2\} \prec_M \{s\}$, the new equational step is smaller than the rewrite steps in the peak.

□

Lemma 3.3.4. *Every proof* Pr *in* (S_*, G_*) *is equivalent to a rewrite proof.*

Proof. By well-founded induction on the well-founded order $\sqsubset$. Let Pr be a proof in (S_*, G_*). If Pr is itself a rewrite proof, then we are done. Otherwise, by **Lemma 3.3.3**, there is a proof Pr$'$ such that Pr$' \sqsubset$ Pr. By induction, Pr$'$, and thus also *Pr*, is equivalent to a rewrite proof. □

Given a polynomial q of the form $c_1m_1 + c_2m_2 + \ldots + c_km_k$, we use

$$s \overset{\langle p,q\rangle}{\leftleftarrows} t$$

to denote a *multi-equational step*, that is, the sequence of equational steps:

$$s \overset{\langle p,c_1m_1\rangle}{\longleftarrow} s_1 \overset{\langle p,c_2m_2\rangle}{\longleftarrow} s_2 \ldots s_{k-1} \overset{\langle p,c_km_k\rangle}{\longleftarrow} t$$

It is easy to see that $t = s - pq$.

Theorem 3.3.5. *Given a set of polynomials* $F = \{p_1, \ldots, p_k\}$*, an arbitrary run*

$$(F = S_1, G_1 = \emptyset) \vdash \ldots \vdash (S_n = \emptyset, G_n)$$

of a correct Gröbner basis procedure $\mathfrak{G}$*, and a polynomial p, the following holds: If* $p \in \mathcal{I}(F)$*, then there exists a rewrite proof for* $p \simeq_F 0$ *using* $\mapsto_{G_n}$*. Moreover,* G_n *is confluent.*

Proof. If $p \in \mathcal{I}(F)$, then we must have $p = p_1q_1 + \ldots + p_kq_k$ for some $q_1, \ldots, q_k \in \mathbb{Q}[\vec{x}]$. Let Pr be the following proof in (S_*, G_*) for $p \simeq_F 0$

$$p \overset{\langle p_1,q_1\rangle}{\leftleftarrows} \ldots \overset{\langle p_k,q_k\rangle}{\leftleftarrows} 0$$

By **Lemma 3.3.4**, Pr is equivalent to a rewrite proof.

Now, we show that G_n is confluent. Suppose not. Let $\mapsto_{G_n}$ be the reduction relation induced by G_n. Since G_n is not confluent, there are polynomials s, t_1 and t_2 such that

$$s \mapsto_{G_n} \ldots \mapsto_{G_n} t_1$$
$$s \mapsto_{G_n} \ldots \mapsto_{G_n} t_2$$

where t_1 and t_2 cannot be reduced by G_n. The reductions above induce a proof Pr in (S_*, G_*) for $t_1 \simeq_F t_2$. Actually, this proof only uses polynomials in G_n, but it has a peak at s. By **Lemma 3.3.4**, there is an equivalent rewrite proof Pr$'$, contradicting the assumption that t_1 and t_2 cannot be reduced by G_n. □

3.4 Criteria for Discarding S-polynomials

Buchberger introduced two criteria for discarding superfluous S-polynomials [Buc79]. We discussed the motivation and importance of such criteria in **Section 2.3.4**. We now examine how these classical criteria can be accommodated in the general setting of Abstract Gröbner Bases. Inspecting the proof of **Lemma 3.3.3**, we see that S-polynomials are only used in **case 3b**, where a non-rewrite proof Pr contains a *peak*. This observation suggests a methodology for proving the strategy-independent admissibility of criteria for discarding redundant S-polynomials.

Observation 3.4.1. An S-polynomial $\mathsf{spol}(p_1, p_2)$ can be discarded if it is not needed to obtain a smaller proof Pr′ in **case 3b** of **Lemma 3.3.3**.

In the following, we assume p_1, p_2 and p_k are polynomials in G_* of the form $\underline{m_1} + q_1$, $\underline{m_2} + q_2$ and $\underline{m_k} + q_k$ respectively. The two criteria we will consider are as follows:

Criterion 1. *If* $\mathsf{lcm}(m_1, m_2) = m_1 m_2$*, then* $\mathsf{spol}(p_1, p_2)$ *is superfluous.*

Criterion 2. *If there exists some* $p_k \in G_*$ *s.t.* $\mathsf{lcm}(m_1, m_2)$ *is a multiple of* m_k *and* $\mathsf{spol}(p_1, p_k)$ *and* $\mathsf{spol}(p_2, p_k)$ *are in* S_**, then* $\mathsf{spol}(p_1, p_2)$ *is superfluous.*

Observation 3.4.2. If $\mathsf{lcm}(m_1, m_2) = m m_k$, then

$$\begin{aligned} \mathsf{lcm}(m_1, m_2) &= (m_{k_1})\mathsf{lcm}(m_1, m_k) \\ \mathsf{lcm}(m_1, m_2) &= (m_{k_2})\mathsf{lcm}(m_2, m_k) \end{aligned}$$

for some m_{k_1} and m_{k_2}. Actually,

$$\begin{aligned} m_{k_1} &= \frac{\mathsf{lcm}(m_1, m_2)}{\mathsf{lcm}(m_1, m_k)}, \\ m_{k_2} &= \frac{\mathsf{lcm}(m_1, m_2)}{\mathsf{lcm}(m_2, m_k)}. \end{aligned}$$

Note that m_{k_1} and m_{k_2} are well defined monomials because $\mathsf{lcm}(m_1, m_2) = \mathsf{lcm}(m_1, m_2, m_k)$.

We first adjust our notion of a correct procedure to take into account the fact that the Superpose rule may be enhanced to carry a side-condition, φ, barring its application.

Definition 3.4.3 (Conditionally Correct Procedure)**.** *A Gröbner basis procedure* $\mathfrak{G}$ *is said to be conditionally* φ*-correct iff it produces only finite runs* $(S_1,\ G_1 = \emptyset) \vdash \ldots \vdash (S_n = \emptyset,\ G_n)$*, and*

$$\mathsf{SP}_\varphi(G_n) \subseteq (S_1 \cup S_2 \cup \ldots \cup S_{n-1}),$$

where

$$\mathsf{SP}_\varphi(G_n) = \{spol(p_1,p_2) \mid p_1,p_2 \in G_n \ \wedge\ \neg\varphi(p_1,p_2)\}.$$

Theorem 3.4.4. *Let* φ_1,φ_2 *be the natural side-conditions barring applications of* Superpose *corresponding to* ***Criteria 1*** *and* ***2*** *respectively. Let* $\mathfrak{G}$ *be a Gröbner basis procedure that is conditionally* $(\varphi_1 \vee \varphi_2)$*-correct. Then,* ***Lemma 3.3.3*** *still holds for* $\mathfrak{G}$*.*

Proof. Inspecting the proof of **Lemma 3.3.3**, it is easy to see that **case 3b** is the only one affected by the restricted Superpose rule. That is, Pr has a peak of the form:

$$t_1 \xleftarrow{\langle p_1, m_1' \rangle} s \xrightarrow{\langle p_2, m_2' \rangle} t_2$$

for p_1 and p_2 in G_n, p_1 and p_2 are of the form $\underline{m_1} + q_1$ and $\underline{m_2} + q_2$ respectively, and $m_1' m_1 = m_2' m_2$. Then, s must be of the form $q_s + cm\tau_{1,2}$, where $\tau_{1,2} = \mathsf{lcm}(m_1,m_2)$. Moreover, we must have:

$$\begin{aligned} t_1 &= q_s - cm\frac{\tau_{1,2}}{m_1}q_1, \\ t_2 &= q_s - cm\frac{\tau_{1,2}}{m_2}q_2. \end{aligned}$$

Now, assume $\mathsf{spol}(p_1,p_2)$ is not in S_* because one of the criteria above was used.

1. Assume $\mathsf{spol}(p_1,p_2)$ is not in S_* because of **Criterion 1**. Then, $\tau_{1,2} = m_1 m_2$, and consequently

$$\begin{aligned} s &= q_s + cmm_1m_2, \\ t_1 &= q_s - cmm_2q_1, \\ t_2 &= q_s - cmm_1q_2. \end{aligned}$$

Now, let s' be the polynomial $q_s + (cm)q_1q_2$, and Pr' be the proof that is obtained by replacing the peak with:

$$t_1 \xleftarrow{\langle p_2, -cmq_1 \rangle}\!\!\!\!\!\rightarrow s' \xleftarrow{\langle p_1, cmq_2 \rangle}\!\!\!\!\!\rightarrow t_2$$

Since, t_1, t_2, s' and every intermediate polynomial in the multi-equational steps above is smaller than s, the new equational steps in Pr' are smaller than the two rewrite rules in the peak in Pr. Therefore, $\mathrm{Pr}' \sqsubset \mathrm{Pr}$.

2. Assume $\mathsf{spol}(p_1, p_2)$ is not in S_* because of **Criterion 2**. Then, there is a p_k of the form $\underline{m_k} + q_k$ in G_* such that $\mathsf{spol}(p_1, p_k)$ and $\mathsf{spol}(p_2, p_k)$ are in S_*, and $\tau_{1,2} = m' m_k$ for some m'. Let $\tau_{1,k} = \mathsf{lcm}(m_1, m_k)$ and $\tau_{2,k} = \mathsf{lcm}(m_2, m_k)$. Then, by **Observation 3.4.2**, we have $\tau_{1,2} = m_{k_1}\tau_{1,k}$ and $\tau_{1,2} = m_{k_2}\tau_{2,k}$.

$$\begin{aligned} t_1 &= q_s - cm\frac{\tau_{1,2}}{m_1}q_1 \\ &= q_s - cm\frac{m_{k_1}\tau_{1,k}}{m_1}q_1 \\ &= q_s - cmm_{k_1}\frac{\tau_{1,k}}{m_1}q_1. \end{aligned}$$

Similarly, $t_2 = q_s - cmm_{k_2}\frac{\tau_{2,k}}{m_2}q_2$. Recall that,

$$\begin{aligned} \mathsf{spol}(p_1, p_k) &= (\frac{\tau_{1,k}}{m_1})q_1 - (\frac{\tau_{1,k}}{m_k})q_k, \\ \mathsf{spol}(p_2, p_k) &= (\frac{\tau_{2,k}}{m_2})q_2 - (\frac{\tau_{2,k}}{m_k})q_k. \end{aligned}$$

Now, let s' be the polynomial $q_s - cm\frac{\tau_{1,2}}{m_k}q_k$. By algebraic manipulation, we have:

$$\begin{aligned} t_1 + cmm_{k_1}\mathsf{spol}(p_1, p_k) &= q_s - cmm_{k_1}\frac{\tau_{1,k}}{m_k}q_k \\ &= q_s - cm\frac{m_{k_1}\tau_{1,k}}{m_k}q_k \\ &= q_s - cm\frac{\tau_{1,2}}{m_k}q_k \\ &= s' \\ &= q_s - cm\frac{m_{k_2}\tau_{2,k}}{m_k}q_k \\ &= q_s - cmm_{k_2}\frac{\tau_{2,k}}{m_k}q_k \\ &= t_2 + cmm_{k_2}\mathsf{spol}(p_2, p_k). \end{aligned}$$

Note that in the equations above, all "fractions" of the form $\frac{m_i}{m_j}$ are actual monomials because in all cases m_j divides m_i. For instance, $\frac{\tau_{1,k}}{m_k}$ is a monomial because m_k always divides $\mathsf{lcm}(m_1, m_k) = \tau_{1,k}$. Now, let Pr' be the proof that is obtained by replacing the peak in Pr with

$$t_1 \xleftarrow{\langle \mathsf{spol}(p_1,p_k), -cmm_{k_1} \rangle} s' \xleftarrow{\langle \mathsf{spol}(p_2,p_k), cmm_{k_2} \rangle} t_2$$

Since t_1, t_2 and s' are smaller than s, we have $\mathrm{Pr}' \sqsubset \mathrm{Pr}$.

□

Definition 3.4.5 (Eager SH-simplification)**.** *We say a Gröbner basis procedure $\mathfrak{G}$ implements eager SH-simplification iff $\mathfrak{G}$ only applies* Orient *to $p \in S_i$ when* Simplify-S *cannot be applied to p, and $\mathfrak{G}$ only attempts*[1] *to apply* Superpose *to $p_1, p_2 \in G_i$ when* Simplify-H *cannot be applied to p_1, p_2.*

Criterion 3. *Assume p_1 and p_2 are polynomials in G_* of the form $\underline{m_1} + q_1$, $\underline{m_2} + q_2$ respectively. If m_1 divides m_2 or m_2 divides m_1, then $\mathsf{spol}(p_1, p_2)$ is superfluous.*

Observation 3.4.6. As a helpful referee of the paper version of this chapter pointed out, it is perhaps unlikely that **Criterion 3** will be very effective in practice when it is combined with the Gebauer-Möller criteria [CKR04]. Nevertheless, in the absence of Gebauer-Möller (we have not examined the admissibility of Gebauer-Möller w.r.t. Abstract GBs), we have found **Criterion 3** to be a very useful component of the term indexing routines used in the implementation of a new class of Gröbner basis algorithms we present in **Chapter 5**. Moreover, we find it to be an interesting example of the usefulness of **Observation 3.4.1** as the basis of a methodology for proving the strategy-independent correctness of superfluous S-polynomial criteria.

Theorem 3.4.7. *Let φ be the natural side-condition for* Superpose *corresponding to **Criterion 3**. Let $\mathfrak{G}$ be a conditionally φ-correct Gröbner basis procedure using eager SH-simplification. Let $\mathfrak{G}$ have the property that it attempts to apply* Superpose *to every $p_1, p_2 \in G_n$. Then, **Lemma 3.3.3** still holds.*

Proof. As in the proof of **Theorem 3.4.4**, we only need to consider **case 3b**. That is, Pr has a peak of the form:

$$t_1 \xleftarrow{\langle p_1, m_1' \rangle} s \xrightarrow{\langle p_2, m_2' \rangle} t_2$$

for p_1 and p_2 in G_n, and p_1 and p_2 are of the form $\underline{m_1} + q_1$ and $\underline{m_2} + q_2$. Now, assume $\mathsf{spol}(p_1, p_2)$ is not in S_* because of **Criterion 3**, then m_1 divides m_2 or m_2 divides m_1. Since $\mathfrak{G}$ uses eager SH-simplification, by **Observation 3.2.6**, $m_1 \neq m_2$. Therefore, m_1 properly divides m_2 or m_2 properly divides m_1. Without loss of generality, assume m_1 properly divides m_2, then p_2 cannot be in G_n because rule Simplify-H would simplify it using p_1. □

[1] By "attempts to apply" we mean that Superpose is either applied as usual, or it is tried but is ultimately skipped because of an active side-condition φ barring its application.

3.5 Conclusion

In conclusion, we have presented an abstract theory of Gröbner basis procedures and used it to prove the strategy-independent admissibility of three superfluous S-polynomial criteria. To accomplish this, we introduced a generalisation of the notion of S-polynomial superfluousness in terms of proof orders. From this generalisation, we derived a methodology for analysing the strategy-independent admissibility of S-polynomial criteria and used it successfully on the three criteria considered.

Chapter 4

Locally Minimal Nullstellensatz Proofs

4.1 Introduction

Recall that Hilbert's Weak Nullstellensatz (**Theorem 2.3.21**) guarantees the existence of ideal membership identities certifying the unsatisfiability of systems of polynomial equations whose corresponding affine varieties over $\mathbb{C}^n$ are empty. In particular, if a system of polynomial equations

$$\bigwedge_{i=1}^{k}(p_i = 0) \qquad (p_i \in \mathbb{Q}[\vec{x}])$$

is unsatisfiable over $\mathbb{C}^n$, then $1 \in \mathcal{I}(\{p_1,\ldots,p_k\})$ and thus $\exists\, q_1,\ldots,q_k \in \mathbb{Q}[\vec{x}]$ s.t.

$$1 = \sum_{i=1}^{k} q_i p_i.$$

Such an ideal membership identity can be considered a *proof* of the complex unsatisfiability of the corresponding polynomial system. This proof shows us that

$$\mathbf{ACF}_0 \models \left(\left(\exists \vec{x} \bigwedge_{i=1}^{k} p_i = 0\right) \iff 0 = 1\right)$$

and, informally, it might be written as follows:

> Assume $\bigwedge_{i=1}^{k}(p_i = 0)$. Then, it must be the case that $\left(\sum_{i=1}^{k} q_i p_i\right) = 0$. But, by simple ring-theoretic reasoning (i.e., polynomial simplification) we see that $\left(\sum_{i=1}^{k} q_i p_i\right) = 1$. Thus, our assumption implies $0 = 1$ and so it cannot be the case that $\bigwedge_{i=1}^{k}(p_i = 0)$.

Given $p_1,\ldots,p_k$ as above, this type of unsatisfiability proof may be succinctly represented simply by recording the polynomials $q_1,\ldots,q_k$. This tuple $\langle q_1,\ldots,q_k\rangle$,

called a tuple of *cofactors* of 1 w.r.t. $p_1, \ldots, p_k$, can be seen as a *proof object*, i.e., a compact representation of a proof certifying the unsatisfiability of $\bigwedge_{i=1}^{k}(p_i = 0)$.

Unsatisfiability proofs as above (and hence their corresponding proof objects) may, however, contain redundant information: a proper subset of the equational assumptions $\{(p_i = 0) \mid 1 \leq i \leq k\}$ used in these proofs may be sufficient to derive the unsatisfiability of the original polynomial system. One trivial way this can happen, for instance, is if some q_i is 0. Then, the corresponding p_i plays no essential role in the fact that $\left(\sum_{i=1}^{k} q_i p_i\right) = 1$, and it may be useful to know that the smaller system of equations

$$p_1 = 0 \wedge \ldots \wedge\ p_{i-1} = 0 \wedge\ p_{i+1} = 0 \wedge \ldots \wedge p_k = 0$$

is actually unsatisfiable as well. Similar phenomena may happen in algebraically non-trivial ways, even when some q_i is not explicitly zero.

For using Nullstellensatz techniques in SMT-based decision methods, a *minimal* proof is often desired, one in which all assumptions (i.e., all p_i) play a vital role. With this in mind, we introduce a notion of *locally* minimal Nullstellensatz proofs and give ideal-theoretic methods for their construction.

4.1.1 Motivation

Modern SMT solvers have application in the verification of software and hardware artifacts and are seeing increasing use in areas as diverse as planning and formalised mathematics. At a high-level, an SMT solver consists of an orchestrated combination of a DPLL based SAT solver and a number of satellite "theory" solvers (T-solvers) which implement decision methods for decidable elementary theories such as linear integer and real arithmetic, bit-vector arithmetic, and the theory of uninterpreted functions with equality. The effectiveness of an SMT decision loop depends crucially upon the ability of its T-solvers to identify "small" inconsistent components of formulas [dMRS04, NO07]. Thus when one develops a new T-solver, the investigation of techniques for finding such "small" inconsistent subformulas is an important concern.

The work described in this chapter can be seen as a contribution to the development of effective T-solvers for nonlinear polynomial arithmetic over both the real and complex numbers. In particular, we consider the problem of finding "small" proof objects certifying the unsatisfiability of systems of polynomial equations over any algebraically closed field. We consider this problem within the context of Gröbner basis calculations.

We start by defining algebraic notions of proof minimality and redundancy. Then, we examine how Gröbner basis procedures can be augmented to produce proof objects certifying their membership judgments. Given these certificates, we introduce two proof minimisation transformations — *cofactor-subsumption* and *basis-subsumption* — for removing redundancy from extracted proof objects. Finally, we illustrate how a restricted form of cofactor subsumption can be efficiently implemented and used to reduce proof redundancy.

4.1.2 Our Contribution

The results of this chapter were obtained jointly with Dr. Leonardo de Moura. The goal of developing algebraic machinery for the minimisation of Nullstellensatz proofs was proposed by Dr. de Moura. We then proved the theorems together and both contributed equally to the work that follows. The results of this chapter were published as [dMP09].

4.2 Algebraic Notions of Proof Minimality

Let $B = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\vec{x}]$. If $p \in \mathcal{I}(B)$ as witnessed by the fact that $\left(\sum_{i=1}^{k} q_i p_i\right) = p$, then we will call $\langle q_1, \ldots, q_k \rangle$ an *ideal membership certificate* (*certificate* for short) showing that $p \in \mathcal{I}(B)$. If $C = \langle q_1, \ldots, q_k \rangle$ is a certificate for the fact that $1 \in \mathcal{I}(B)$, then we will call C a *Nullstellensatz proof* (*proof* for short) showing that $1 \in \mathcal{I}(B)$. When convenient, we may also call C a *proof* for the *unsatisfiability of* B. What we mean in this case is that C bears witness to the fact that the equational system $\bigwedge_{i=1}^{k}(p_i = 0)$ is unsatisfiable over $\mathbb{C}^n$.

4.2.1 Algebraic Notions of Redundancy

Definition 4.2.1 (Basis redundancy)**.** *We say B is* p-non-redundant *iff*

$$p \in \mathcal{I}(B) \ \wedge \ \forall B' \subset B \, (p \notin \mathcal{I}(B)).$$

Similarly, we say B is p-redundant *iff*

$$p \in \mathcal{I}(B) \ \wedge \ \exists B' \subset B \, (p \in \mathcal{I}(B)).$$

Definition 4.2.2 (Membership set)**.** *We define*

$$Mem(p, p_1, \ldots p_k) \subseteq (\mathbb{Q}[\vec{x}])^k$$

to be the collection of ideal membership certificates showing $p \in \mathcal{I}(\{p_1,\ldots,p_k\})$ *as follows:*

$$Mem(p,p_1,\ldots,p_k) = \left\{ \langle q_1,\ldots,q_k\rangle \mid \sum_{i=1}^{k} p_i q_i = p \right\}.$$

We may write $Mem(p,B)$ *in place of* $Mem(p,p_1,\ldots p_k)$. *Given* $\alpha \in Mem(p,B)$, *coordinate* $\alpha(i)$ *is known as the ith cofactor (of p w.r.t. B) in* α.

Definition 4.2.3 (Proof set)**.** *We define* $\Pr(p_1,\ldots p_k)$ *to be the collection of Nullstellensatz proofs of the unsatisfiability of* $\{p_1,\ldots,p_k\}$[12] *over* $\mathbb{C}^n$. *That is,*

$$\Pr(p_1,\ldots p_k) = Mem(1,p_1,\ldots p_k).$$

We may write $\Pr(B)$ *in place of* $\Pr(p_1,\ldots p_k)$.

Given a certificate $\alpha \in Mem(p,B)$, let us call those members of B whose corresponding cofactors in α are non-zero the *hypotheses* used in α.

Definition 4.2.4 (Basis of hypotheses)**.** *Given an ideal membership certificate* $\alpha \in Mem(p,B)$, *we define* $Hyp(B,\alpha)$ *to be the collection of B-hypotheses used in* α *as follows:*

$$Hyp(B,\alpha) = \{p_i \in B \mid \alpha(i) \neq 0\}.$$

Then, there is a succinct account of what it means for a certificate to be *non-redundant*.

Definition 4.2.5 (Non-redundant certificates)**.** *We say a certificate* $\alpha \in Mem(p,B)$ *is* non-redundant *iff* $Hyp(B,\alpha)$ *is p-non-redundant.*

Observe that $\alpha \in Mem(p,B)$ is non-redundant iff

$$\neg\exists \alpha' \in Mem(p,B) \text{ s.t. } Hyp(B,\alpha') \subset Hyp(B,\alpha).$$

Restricted to proofs, this means $\alpha \in \Pr(B)$ is non-redundant iff

$$\neg\exists \alpha' \in Pr(B) \text{ s.t. } Hyp(B,\alpha') \subset Hyp(B,\alpha).$$

Thus if $\alpha \in \Pr(B)$ is a non-redundant proof, then no strict subset of the hypotheses used in the proof is sufficient to show the unsatisfiability of B. However, this is an

[1]The interested reader may note the connection between $\Pr(p_1,\ldots,p_k)$ and the first syzygy module of $\langle p_1,\ldots,p_k\rangle$. In particular, $Syz(p_1,\ldots,p_k) = Mem(0,p_1,\ldots,p_k)$ while $\Pr(p_1,\ldots,p_k) = Mem(1,p_1,\ldots,p_k)$.

[2]Recall our convention that by Nullstellensatz proof for the "unsatisfiability of $\{p_1,\ldots,p_k\}$," we mean for the unsatisfiability of the associated equational system $\bigwedge_{i=1}^{k}(p_i = 0)$.

essentially *local* notion, dependent on the context of the current proof. In particular, the non-redundancy of a proof α does not in general mean that there is no smaller subset $B' \subset B$ s.t. $|B'| < |Hyp(B,\alpha)|$ that is itself unsatisfiable. This can be seen with the following simple example.

Example 4.2.6. *Let the system* Γ *of polynomial equations be defined as follows:*

$$\Gamma = \{x^2y^2 - 1 = 0,\ x^2y = 0,\ xy = 0,\ x+1 = 0,\ y+1 = 0\}.$$

Let $B = \{x^2y^2 - 1, x^2y, xy, x+1, y+1\}$ *be the basis of polynomials corresponding to* Γ. *Observe that* $\mathrm{Pr}(B) \neq \emptyset$. *Among others, it contains the following two proofs:*

$$\alpha = \langle -1, y, 0, 0, 0\rangle \text{ for } 1 = (-1)(x^2y^2 - 1) + y(x^2y), \text{ and}$$
$$\beta = \langle 0, 0, 1, -y, 1\rangle \text{ for } 1 = xy + -y(x+1) + y + 1.$$

Then, we have, $Hyp(B,\alpha) = \{x^2y^2 - 1, x^2y\}$, $Hyp(B,\beta) = \{xy, x+1, y+1\}$. *Observe that both* $Hyp(B,\alpha)$ *and* $Hyp(B,\beta)$ *are non-redundant and* $|Hyp(B,\alpha)| < |Hyp(B,\beta)|$.

Thus, non-redundancy of a proof does not mean it is a proof that uses the *globally* least number of hypotheses, but rather that it is in some sense *locally* minimal: If one begins with a non-redundant proof and drops any used hypotheses, then no proof of unsatisfiability for the resulting system will exist. This is made precise with the following lemma whose proof is immediate.

Lemma 4.2.7. *Let* $\alpha \in \mathrm{Pr}(B)$ *be a non-redundant proof. Then, every* $B' \subset Hyp(B,\alpha)$ *is satisfiable over* $\mathbb{C}^n$.

4.3 Extracting Certificates from GB Procedures

Before investigating how certificates may be minimised, we need to make clear how a Gröbner basis procedure can be used to construct them. We will do this by showing how one can extend the Abstract Gröbner Bases calculus in **Figure 3.1** so that the rules allow simple certificate extraction. We first present a naive approach and then give a refinement of it which helps facilitate structure sharing in an implementation.

Let us fix a bit of notation. If $\alpha, \beta \in (\mathbb{Q}[\vec{x}])^k$, and p in $\mathbb{Q}[\vec{x}]$, then $\alpha + \beta$ will denote $\langle \alpha(1) + \beta(1), \ldots, \alpha(k) + \beta(k)\rangle$, and $p\alpha$ will denote $\langle p\alpha(1), \ldots, p\alpha(k)\rangle$. As in the previous chapter, in the context of polynomials $p_1 = \underline{m_1} + q_1$ and $p_2 = \underline{m_2} + q_2$, we will use $\tau_{1,2}$ to denote $\mathsf{lcm}(m_1, m_2)$.

Definition 4.3.1 (Certified polynomial)**.** A *certified polynomial* (w.r.t. B) is a pair (p,α) s.t. $\alpha \in \mathrm{Mem}(p,B)$.

Figure 4.1 contains the rules of **Figure 3.1** lifted to certified polynomials.
We use $\mathbf{1}_j$ to denote $\langle q_1,\ldots,q_k\rangle \in (\mathbb{Q}[\vec{x}])^k$, where $q_j = 1$, and $q_i = 0$ for all $j \neq i$.

Definition 4.3.2 (Certified Procedure)**.** *A certified Gröbner basis procedure $\mathfrak{G}$ is a program that accepts a set of polynomials $\{p_1,\ldots,p_k\}$, a monomial order $\prec$, and uses the rules of **Figure 4.1** to generate a (finite or infinite) sequence*

$$(S_1 = \{(p_1,\mathbf{1}_1),\ldots,(p_k,\mathbf{1}_k)\},\ G_1 = \emptyset) \vdash (S_2,G_2) \vdash (S_3,G_3) \vdash \ldots$$

Note that if $(1,\alpha) \in S_i$ for some i, then α is a proof for the complex unsatisfiability of $\bigwedge_{i=1}^{k}(p_i = 0)$.

In the linear case, *zero variables* can be used to represent certified polynomials without having to introduce any extra machinery like we have with the "lifted" inference rules. This has been investigated within the context of both simplex and Gaussian elimination [AB98, RS04]. The idea is to represent the certified polynomial (p,α) as $p - \alpha(1)z_1 - \ldots - \alpha(k)z_k$, where the z_i's are new fresh variables. Then, the coefficients of the variables z_i are used to track the the certificate coordinates $\alpha(i)$ as one takes linear combinations of the polynomials. The new polynomial is still linear because $\alpha(i)$ is always a constant for the linear case. From the Gröbner basis perspective, an approach based on zero variables is attractive because a regular Gröbner basis procedure, i.e., one using the rules in **Figure 3.1**, could be used to obtain certificates. To do so, however, one need make the zero variables z_i smaller than the variables $\{x_1,\ldots,x_n\}$. This is so that the z_i are not eliminated during Gröbner basis construction. This approach cannot be directly applied to the nonlinear case, because it would require us to make any monomial containing a zero variable z_i smaller than a monomial not containing any zero variable. There is no monomial order with such property, because it violates admissibility. For example, it would require $z_2x_1 \prec x_1$. Thus, an approach like ours seems necessary when certificate extraction is needed over nonlinear systems.

4.3.1 Structured Certificates

The overhead in a certified Gröbner basis procedure is substantial, since the certificates α can grow in size very quickly. Moreover, it wasteful to compute a certificate for a polynomial that is deleted using the Delete rule. We address this issue with *structured*

Orient

$$\frac{S \cup \{(\underline{cm} + q, \alpha)\}, G}{S, G \cup \{(\underline{m} + (\frac{1}{c})q, (\frac{1}{c})\alpha)\}}$$

Superpose

$$\frac{S, G \cup \{(\overbrace{\underline{m_1} + q_1}^{p_1}, \alpha_1), (\overbrace{\underline{m_2} + q_2}^{p_2}, \alpha_2)\}}{S \cup \{(\mathsf{spol}(p_1, p_2), (\frac{\tau_{1,2}}{m_1})\alpha_1 - (\frac{\tau_{1,2}}{m_2})\alpha_2)\}, G \cup \{(p_1, \alpha_1), (p_2, \alpha_2)\}}$$

Delete

$$\frac{S \cup \{(0, \alpha)\},\ G}{S, G}$$

Simplify-S

$$\frac{S \cup \{(c_1 m_1 m_2 + q_1,\ \alpha_1)\}, G \cup \{(\underline{m_2} + q_2,\ \alpha_2)\}}{S \cup \{(q_1 - c_1 m_1 q_2,\ \alpha_1 - c_1 m_1 \alpha_2\}, G \cup \{(\underline{m_2} + q_2,\ \alpha_2)\}}$$

Simplify-H

$$\frac{S, G \cup \{(\underline{m_1 m_2} + q_1, \alpha_1),\ (\underline{m_2} + q_2, \alpha_2)\}}{S \cup \{(q_1 - m_1 q_2, \alpha_1 - m_1 \alpha_2)\}, G \cup \{(\underline{m_2} + q_2, \alpha_2)\}}$$

if $m_1 \neq 1$

Simplify-T

$$\frac{S, G \cup \{(\underline{m} + c_1 m_1 m_2 + q_1, \alpha_1),\ (\underline{m_2} + q_2, \alpha_2)\}}{S, G \cup \{(\underline{m} - c_1 m_1 q_2 + q_1, \alpha_1 - c_1 m_1 \alpha_2),\ (\underline{m_2} + q_2, \alpha_2)\}}$$

Figure 4.1: Lifted inference rules.

certificates. Structured certificates are represented using the constructors A (assumption), S (superpose), R (simplify), D (divide).

Definition 4.3.3 (Structured Certificates)**.** *The set of polynomial structured certificates, $\mathcal{C}$, is defined as the least set s.t.*

Assert: $p \in \mathbb{Q}[\vec{x}] \Longrightarrow \mathsf{A}(p) \in \mathcal{C}$,

Superpose: $\varphi_1, \varphi_2 \in \mathcal{C} \Longrightarrow \mathsf{S}(\varphi_1, \varphi_2) \in \mathcal{C}$,

Simplify: $\varphi_1, \varphi_2 \in \mathcal{C} \ \wedge \ m \in \mathbb{M} \Longrightarrow \mathsf{R}(\varphi_1, \varphi_2, m) \in \mathcal{C}$,

Divide: $\varphi \in \mathcal{C} \Longrightarrow \mathsf{D}(\varphi) \in \mathcal{C}$.

Figure 4.2 contains the lifted rules using structured certificates. The initial state (S_1, G_1) for a procedure using structured certificates is:

$$(\{(p_1,\ \mathsf{A}(p_1)), \ldots,\ (p_k,\ \mathsf{A}(p_k))\},\ \emptyset).$$

The set of hypotheses $hyp(\varphi)$ of a structured certificate φ is defined as:

$$\begin{aligned} hyp(\mathsf{A}(p)) &= p, \\ hyp(\mathsf{S}(\varphi_1, \varphi_2)) &= hyp(\varphi_1) \cup hyp(\varphi_2), \\ hyp(\mathsf{R}(\varphi_1, \varphi_2, m)) &= hyp(\varphi_1) \cup hyp(\varphi_2), \\ hyp(\mathsf{D}(\varphi)) &= hyp(\varphi). \end{aligned}$$

Definition 4.3.4 (Polynomial of a Certificate)**.** *Given a structured certificate $\varphi \in \mathcal{C}$, the polynomial of φ, $pol(\varphi)$, is defined as follows:*

1. $pol(\mathsf{A}(p)) = p$.
2. $pol(\mathsf{S}(\varphi_1, \varphi_2)) = \mathsf{spol}(pol(\varphi_1), pol(\varphi_2))$.
3. $$pol(\mathsf{R}(\varphi_1, \varphi_2, m)) = \begin{cases} q_1 - c_1 m_1 q_2 \\ \quad \textit{if } pol(\varphi_1) \textit{ contains } m \\ \quad \textit{where} \\ \qquad pol(\varphi_1) = c_1 m_1 m_2 + q_1, \\ \qquad m = m_1 m_2, \\ \qquad pol(\varphi_2) = \underline{m_2} + q_2 \\ pol(\varphi_1) \ \ \textit{otherwise.} \end{cases}$$

Orient

$$\frac{S \cup \{(\underline{cm} + q, \varphi)\}, G}{S, G \cup \{(\underline{m} + (\frac{1}{c})q, \mathsf{D}(\varphi))\}}$$

Superpose

$$\frac{S, G \cup \{(p_1, \varphi_1), (p_2, \varphi_2)\}}{S \cup \{(\mathsf{spol}(p_1, p_2), \mathsf{S}(\varphi_1, \varphi_2)\}, G \cup \{(p_1, \varphi_1), (p_2, \varphi_2)\}}$$

Delete

$$\frac{S \cup \{(0, \varphi)\},\ G}{S, G}$$

Simplify-S

$$\frac{S \cup \{(c_1 m_1 m_2 + q_1,\ \varphi_1)\}, G \cup \{(\underline{m_2} + q_2,\ \varphi_2)\}}{S \cup \{(q_1 - c_1 m_1 q_2,\ \mathsf{R}(\varphi_1, \varphi_2, m_1 m_2))\}, G \cup \{(\underline{m_2} + q_2,\ \varphi_2)\}}$$

Simplify-H

$$\frac{S, G \cup \{(\underline{m_1 m_2} + q_1, \varphi_1),\ (\underline{m_2} + q_2, \varphi_2)\}}{S \cup \{(q_1 - m_1 q_2, \mathsf{R}(\varphi_1, \varphi_2, m_1 m_2))\}, G \cup \{(\underline{m_2} + q_2, \varphi_2)\}}$$

if $m_1 \neq 1$

Simplify-T

$$\frac{S, G \cup \{(\underline{m} + c_1 m_1 m_2 + q_1, \varphi_1),\ (\underline{m_2} + q_2, \varphi_2)\}}{S, G \cup \{(\underline{m} - c_1 m_1 q_2 + q_1, \mathsf{R}(\varphi_1, \varphi_2, m_1 m_2)),\ (\underline{m_2} + q_2, \varphi_2)\}}$$

Figure 4.2: Lifted inference rules with structured certificates.

4. $pol(\mathsf{D}(\varphi)) = m + (\frac{1}{c})q$, *if* $pol(\varphi) = \underline{cm} + q$.

Observation 4.3.5. If a structurally certified polynomial (p, φ) appears during a run of a structually certifying Gröbner basis procedure, then $p = pol(\varphi)$.

Definition 4.3.6 (Flat Certificates)**.** *Given a structured certificate* $\varphi \in \mathcal{C}$*, where* $hyp(\varphi) \subseteq B = \{p_1, \ldots, p_k\}$*, the flat certificate with respect to B,* $flat(\varphi)$*, is defined as follows:*

1. $flat(\mathsf{A}(p_i)) = \mathbf{1}_i$.

2. $flat(\mathsf{S}(\varphi_1, \varphi_2)) = (\frac{\tau_{1,2}}{m_1})(flat(\varphi_1)) - (\frac{\tau_{1,2}}{m_2})(flat(\varphi_2))$,
   where $pol(\varphi_1) = \underline{m_1} + q_1$, $pol(\varphi_2) = \underline{m_2} + q_2$, *and* $\tau_{1,2} = lcm(m_1, m_2)$.

3. $$flat(\mathsf{R}(\varphi_1, \varphi_2, m)) = \begin{cases} flat(\varphi_1) - c_1 m_1 (flat(\varphi_2)) \\ \quad \textit{if } pol(\varphi_1) \textit{ contains } m, \\ \quad \textit{where} \\ \quad\quad pol(\varphi_1) = c_1 m_1 m_2 + q_1, \\ \quad\quad m = m_1 m_2, \\ \quad\quad pol(\varphi_2) = \underline{m_2} + q_2 \\ flat(\varphi_1) \quad \textit{otherwise.} \end{cases}$$

4. $flat(\mathsf{D}(\varphi)) = \frac{1}{c}(flat(\varphi))$, *where* $pol(\varphi) = \underline{cm} + q$.

Theorem 4.3.7. *Given* $B = \{p_1, \ldots, p_k\}$*, and a certificate* $\varphi \in \mathcal{C}$ *where* $hyp(\varphi) \subseteq B$*, then* $flat(\varphi) \in Mem(pol(\varphi), B)$.

4.4 Redundancy

We now wish to address the following fundamental problem: Given a certificate $\alpha \in Mem(p, B)$, can α be feasibly transformed into a non-redundant certificate? With feasibility in mind, we look only for transformations which arise by a combination of (i) dropping used hypotheses and (ii) modifying non-zero cofactors. In particular, all transformations $\alpha \mapsto \alpha'$ are s.t. $Hyp(B, \alpha') \subset Hyp(B, \alpha)$.

In devising techniques to eliminate redundancy, we will need to refer to individual hypotheses contributing to the redundancy.

Definition 4.4.1. *Given a certificate* $\alpha \in Mem(p, B)$ *and a* j *s.t.* $1 \leq j \leq k$*, we say* α *is* j*-redundant iff*

$$\alpha(j) \neq 0 \ \wedge \ Mem(p, Hyp(B, \alpha) \setminus \{p_j\}) \neq \emptyset.$$

4.4.1 Redundancy in the General Case

We now turn to proof redundancy in the context of the general nonlinear case. The following concepts form the basis for our proof minimisation transformations.

Definition 4.4.2. *Given a certificate* $\alpha \in Mem(p,B)$ *and a* j *s.t.* $1 \leq j \leq k$. *Let* H_j *be the set* $Hyp(B,\alpha) \setminus \{p_j\}$. *We say* α *is*

- *j-cofactor-subsumed* $\iff \alpha(j) \in \mathcal{I}(H_j)$,
- *j-basis-subsumed* $\iff p_j \in \mathcal{I}(H_j)$,
- *j-*$\star$*-subsumed* $\iff \alpha(j)p_j \in \mathcal{I}(H_j)$.

First, we focus on cofactor-subsumption. Note that j-cofactor-subsumption is an algebraic generalisation – using the intuition that ideals are an algebraic generalisation of zeroness – of the fact that if a cofactor coordinate $\alpha(j)$ of a certificate is explicitly 0, then its corresponding hypothesis p_j does not contribute to the certificate in an essential way. Let $\alpha \in Mem(p,B)$ and $\beta \in Mem(\alpha(j),B) \neq \emptyset$ with $Hyp(B,\beta) \subseteq Hyp(B,\alpha) \setminus \{p_j\}$. Then, we define the *certificate transformer* $\triangle_{j,\beta}(\alpha)$ for j-cofactor-subsumption (w.r.t. $B = \{p_1,\ldots,p_k\}$) as

$$\triangle_{j,\beta}(\alpha) \;=\; \alpha + (-\alpha(j))\mathbf{1}_j + p_j\beta.$$

Theorem 4.4.3. *Let* $\alpha \in Mem(p,B)$ *be a* j*-cofactor-subsumed certificate with* $Hyp(B,\alpha) = H$, *and* $\beta \in Mem(\alpha(j),B) \neq \emptyset$ *with* $Hyp(B,\beta) \subseteq H \setminus \{p_j\}$. *Then,* $\triangle_{j,\beta}(\alpha) \in Mem(p,B)$, *and* $Hyp(B,\triangle_{j,\beta}(\alpha)) \subseteq H \setminus \{p_j\}$.

The proof of **Theorem 4.4.3** is verified by the following identity.

Proof.

$$
\begin{aligned}
p = \sum_{i=1}^{k} \alpha(i)p_i &= \left(\sum_{i=1}^{j-1} \alpha(i)p_i\right) + \alpha(j)p_j + \left(\sum_{i=j+1}^{k} \alpha(i)p_i\right) \\
&= \left(\sum_{i=1}^{j-1} \alpha(i)p_i\right) + p_j\left(\left(\sum_{i=1}^{j-1} \beta(i)p_i\right) + \left(\sum_{i=j+1}^{k} \beta(i)p_i\right)\right) \\
&\quad + \left(\sum_{i=j+1}^{k} \alpha(i)p_i\right) \\
&= \left(\sum_{i=1}^{j-1} \alpha(i)p_i\right) + \left(\sum_{i=1}^{j-1} p_j\beta(i)p_i\right) + \left(\sum_{i=j+1}^{k} p_j\beta(i)p_i\right) \\
&\quad + \left(\sum_{i=j+1}^{k} \alpha(i)p_i\right) \\
&= \left(\sum_{i=1}^{j-1} \alpha(i)p_i + p_j\beta(i)p_i\right) + \left(\sum_{i=j+1}^{k} \alpha(i)p_i + p_j\beta(i)p_i\right) \\
&= \left(\sum_{i=1}^{j-1} (\alpha(i) + p_j\beta(i))p_i\right) + \left(\sum_{i=j+1}^{k} (\alpha(i) + p_j\beta(i))p_i\right) \\
&= \sum_{i=1}^{k} \left(\triangle_j(\alpha)(i)\right) p_i
\end{aligned}
$$

where $\left[\triangle_j(\alpha)(i) = \left(\triangle_{j,\beta}(\alpha)\right)(i)\right]$. □

Similarly, we define the *certificate transformer* $\bigtriangledown_{j,\beta}(\alpha)$ for j-basis-subsumption (w.r.t. $B = \{p_1, \ldots, p_k\}$) as

$$\bigtriangledown_{j,\beta}(\alpha) = \alpha + (-\alpha(j))\mathbf{1}_j + \alpha(j)\beta.$$

The correctness of this transformer is verified by an algebraic computation analogous to the proof of **Theorem 4.4.3**. Note that, in this case, $\beta \in Mem(p_j, B) \neq \emptyset$.

So, we now have certificate transformers for eliminating the forms of redundancy elucidated by the concepts of j-cofactor and j-basis subsumption. If you recall, however, we also introduced a third related concept, that of j-$\star$-subsumption. It turns out that j-$\star$-subsumption is a more difficult one, and we have not made much progress on it. One useful observation is that if our $\mathcal{I}(B)$ is a *prime* ideal, then j-$\star$-subsumption is actually not needed. This is because in this context a given certificate $\alpha \in Mem(p, B)$ will be j-$\star$-subsumed iff it is either j-cofactor-subsumed or j-basis-subsumed. But, other than the setting of prime ideals, investigating methods for certificate minimisation along j-$\star$-subsumed certificates remains as future work.

4.4.2 Restricted Cofactor-Subsumption and Basis-Subsumption

We use j-subsumption to denote j-cofactor-subsumption and j-basis-subsumption. We now address the following issue: How can j-subsumption be applied effectively in practice? In general, it is too expensive to check whether a certificate α can be j-subsumed or not, because it requires us to answer ideal membership subqueries. That is, given a certificate α, to check whether α can be j-subsumed, we need to compute a Gröbner basis for $Hyp(B,\alpha) \setminus \{p_j\}$. We overcome this difficulty by approximating the ideal membership subqueries. The idea is to answer these queries using a set of rewrite rules that is not necessarily confluent.

Definition 4.4.4 (j-φ-Independent Polynomial)**.** *Given a certificate φ, a certified polynomial (r,φ') is j-φ-independent iff $hyp(\varphi') \subseteq hyp(\varphi) \setminus \{p_j\}$.*

Let $(S_1,G_1) \vdash \ldots \vdash (S_m,G_m)$ be a run produced by a certified Gröbner basis procedure $\mathfrak{G}$, (p,φ) be some certified polynomial in $\cup_{i=1}^{m}(S_i \cup G_i)$, and $\Theta_{j,\varphi}$ be the set of j-φ-independent polynomials in $\cup_{i=1}^{m} G_i$. Now, suppose we want to check whether $\alpha = \mathit{flat}(\varphi)$ is j-cofactor-subsumed or not. Then, we can simply check whether $\alpha(j)$ rewrites to 0 using an arbitrary subset of $\Theta_{j,\varphi}$. For example, in our prototype, we do not track all polynomials produced in a run. Thus, whenever a certified polynomial (c,φ) (with $c \neq 0$) is included in S_m, we use just the j-φ-independent polynomials in G_m (instead of $\cup_{i=1}^{m} G_i$) to check whether $\mathit{flat}(\varphi)$ can be j-cofactor-subsumed or not.

Example 4.4.5. *Let S be a set of polynomials $\{p_1,\ p_2,\ p_3,\ p_4\}$, where:*

$$\begin{aligned} p_1 &= x_1 - x_2, \\ p_2 &= x_1x_3^2 - x_1x_4^2 + 1, \\ p_3 &= x_5x_4 - x_3, \\ p_4 &= x_5x_3 - x_4 \end{aligned}$$

The system $\{p_1 = 0,\ p_2 = 0,\ p_3 = 0,\ p_4 = 0\}$ is unsatisfiable over $\mathbb{C}^5$. Let $\mathfrak{G}$ be a correct structurally certifying Gröbner basis procedure that produces the run $(S_1 = S, G_1 = \emptyset) \vdash \ldots \vdash (S_m,G_m)$, where S_m contains the certified polynomial $(1,\varphi)$, where:

$$\varphi = \mathsf{R}(\mathsf{S}(p_3,p_4),\mathsf{R}(\mathsf{A}(p_1),\mathsf{R}(\mathsf{A}(p_1),\mathsf{A}(p_2),x_3^2),x_4^2),x_2).$$

The flat certificate $\mathit{flat}(\varphi)$ associated with φ is:

$$\mathit{flat}(\varphi) = \langle(-x_3^2 + x_4^2),\ 1,\ x_2x_3,\ -x_2x_4\rangle.$$

Assume also that some G_i in the run contains the certified polynomial $(r, \varphi') = (x_3 - x_4,\ \mathsf{S}(\mathsf{A}(p_3), \mathsf{A}(p_4)))$. Note that (r, φ') is 1*-φ-independent, and $-x_3^2 + x_4^2 \mapsto_r 0$. Thus, flat$(\varphi)$ can be* 1*-cofactor-subsumed.*

4.5 Conclusion

The effectiveness of an SMT solver depends crucially upon the ability of its T-solvers to identify "small" inconsistent sets of formulas. Hence, to address this need in the context of T-solvers for nonlinear arithmetic, we defined algebraic notions of proof minimality and redundancy for complex unsatisfiability proofs based upon Hilbert's weak Nullstellensatz, and introduced two useful certificate transformations aimed at the local minimisation of such proofs: *cofactor-subsumption* and *basis-subsumption*. We also described how ideal membership certificates can be extracted in the framework of Abstract Gröbner Bases. In the next chapter, we will examine in detail a new class of Gröbner basis algorithms tailored to the needs of SMT solvers. These algorithms will be proven correct using Abstract Gröbner Bases and now form the foundation of the nonlinear reasoning techniques present in the Z3 SMT solver [MB08]. Under the hood, these algorithms will make use of the structural certificate machinery we have presented in this chapter.

Chapter 5

Gröbner Basis Algorithms for L3 Nonlinear Systems

5.1 Introduction

In this chapter, we present novel Gröbner basis algorithms based on saturation loops used by modern superposition theorem provers. By combining

- top-level Gröbner basis construction strategies based on the OTTER [McC03] and DISCOUNT [ADF95] saturation loops, and
- term indexing techniques making use of superfluous S-polynomial criteria in Gröbner basis theory,

we are able to compute Gröbner bases for large, largely linear nonlinear systems of polynomial equations which are beyond the reach of previously available methods. These types of systems are typical of those arising from the use of SMT solvers in reasoning about industrial-strength software artifacts with nonlinear arithmetical components. Proving the correctness of these new Gröbner basis procedures is nontrivial, and to do so we utilise the theory of Abstract Gröbner Bases introduced in **Chapter 3**. We illustrate the practical value of the algorithms through an experimental implementation within the Z3 SMT solver [MB08].

5.1.1 Motivation

In attempting to integrate Gröbner basis calculations within Z3, we observed that the known Gröbner basis procedures used for solving difficult algebro-geometric problems

and available in modern computer algebra systems, such as Buchberger's Algorithm [Buc65] and its enhancements F4 and F5 [Fau99, Fau02], were not able to cope with the flavour of large systems of polynomial constraints generated by the SMT solver. These types of nonlinear systems, usually derived from industrial software verification conditions, often contain massive ($> 1,000$, even at times $> 10,000$) numbers of polynomial equations, but have a proportionally small (usually $< 5\%$) nonlinear component. We call these types of systems 'large, largely linear' or 'L3' nonlinear systems. This chapter is focused on the development of novel Gröbner basis calculation algorithms which allow us to compute with these L3 systems.

Tasked with the problem of constructing new Gröbner basis calculation algorithms tailored to the needs of SMT solvers, a very pleasing solution presented itself: We were able to exploit years of work undertaken within the automated theorem proving community and adapt saturation loops and fast term indexing techniques used by modern superposition theorem provers to the context of Gröbner basis calculation.

These loops[1], one derived from McCune's OTTER, the other from Avenhaus et al's DISCOUNT, combined with sophisticated term indexing, have enabled modern high-performance theorem provers to reason effectively in the context of massive clause sets [RV03]. By adapting these developments to a Gröbner basis setting, we are able to leverage work done in one community to aid another. Indeed, these new algorithms allow us to compute Gröbner bases for systems much larger than those amenable to previously available Gröbner basis algorithms, provided that these systems contain a relatively small nonlinear component. While mapping these saturation loops to a Gröbner basis setting is straight-forward, both proving their correctness and deriving appropriate term indexing techniques is not. To prove correctness of the top-level algorithms and justify the term indexing techniques described, we make extensive use of the theory of Abstract Gröbner Bases (cf. **Chapter 3**).

5.1.2 Related Work and Novelty

The idea of using sophisticated simplification and term indexing techniques during Gröbner basis construction has been explored by many, though the latter usually under a different name: as Gröbner basis procedures deal solely with polynomials, the phrase

[1]For a presentation of the OTTER and DISCOUNT loops in the automated theorem proving literature, we suggest (in addition to the original references by McCune [McC03] and Avenhaus et al [ADF95]) the reference [RV03]. As our focus is upon describing and evaluating our Gröbner basis algorithms, we will spend a relatively short amount of time discussing how they are related to the original OTTER and DISCOUNT saturation loops.

"term indexing" is usually eschewed in Gröbner basis research in favor of "polynomial representation" [Car10]. For example, the (very different) techniques underlying both Faugère's F4 [Fau99] and the Cory-Rossin-Salvy "sandpiles" method [CRS02] may be seen as combining sophisticated simplification and term indexing [Car10].

Given that two core ideas explored in this work – using sophisticated simplification and term indexing techniques during Gröbner basis construction – have been explored by many, we find it prudent to make clear which aspects of this work are novel.

Our main contribution is the particular *instantiation* of these ideas. This instantiation has been motivated by the types of problems encountered when using the Z3 SMT solver to verify programs with nonlinear arithmetical components and is particularly interesting from the perspective of automated theorem proving. While a number of prior works have put forth theoretical frameworks for building specialised Gröbner basis procedures, we are aware of none which actually *apply them* and present the details of such a specialisation from algorithm description and correctness to implementation and empirical evaluation. By focusing on specific saturation loops which have been successful in automated theorem proving and mapping them to a Gröbner basis setting, and by undertaking this work in the context of a high-performance SMT solver, we provide a foundation upon which other SMT solver researchers may build. Similarly, we feel this work gives a tangible basis for researchers in automated theorem proving to consider how other techniques in their repertoire may be imported to a Gröbner basis setting.

5.1.3 Our Contribution

The theoretical results of this chapter were obtained jointly with Dr. Leonardo de Moura. The goal of adapting high-performance theorem proving saturation and simplification loops to a Gröbner basis setting was proposed by Dr. de Moura. This goal in fact motivated much of the work presented in **Chapters 3-5**. Dr. de Moura and I both contributed equally to the theoretical work presented. The high-performance implementation of these procedures into a special version of the Z3 SMT solver was completed by Dr. de Moura. The experimental evaluation of these implemented versions of these new procedures in comparison with other Gröbner basis packages was completed by Dr. Paul B. Jackson using a random problem generator written by Dr. de Moura. The results of this chapter were published as [PdMJ10].

5.2 Algorithms: OTTER-GB and DISCOUNT-GB

We now describe two new algorithms for computing Gröbner bases. These algorithms are aimed at solving L3 systems which are beyond the reach of previously available methods. Descriptions of the two algorithms, using the calculus of Abstract Gröbner Bases presented in **Chapter 3**, appear in **Figures 5.1** and **5.2**.

There are two main procedural ingredients to these algorithms: (i) top-level loops adapted from the OTTER [McC03] and DISCOUNT [ADF95] saturation algorithms, and (ii) term indexing techniques derived from both the theorem proving literature [SRV01] and "superfluous S-polynomial criteria" which are important in Gröbner basis theory. The term indexing techniques are designed for facilitating fast applications of the inference rules Superpose, Simplify-S, Simplify-T, and Simplify-H. In **Section 5.2.3**, we will show that these algorithms correspond formally to *correct strategies* in the sense of of Abstract GBs. Thus, by **Theorem 3.2.4**, they will be guaranteed to be terminating, functionally correct Gröbner basis construction algorithms.

5.2.1 Understanding the Algorithms

To help understand these algorithms, it is instructive to examine some differences between them and Buchberger's Algorithm. Before doing so, let us first reflect on how OTTER-GB and DISCOUNT-GB differ from each other.

It is easy to see, by induction, the key difference between the two algorithms: OTTER-GB maintains the invariant that G and S are always maximally simplified w.r.t. G, whereas DISCOUNT-GB maintains only the invariant that G is maximally simplified w.r.t. itself. These differences (with G and S corresponding to the `active` and `passive` sets, respectively) mirrors the key difference between the OTTER and DISCOUNT superposition saturation loops [RV03]. Note that Buchberger's Algorithm maintains neither of these invariants (we will discuss this more shortly). Observe also that both OTTER-GB and DISCOUNT-GB implement eager SH-simplification (cf. **Definition 3.4.5**).

Note that these invariants for our Gröbner basis algorithms lead to nontrivial dynamics in the way polynomials are moved between G and S. For instance, whenever the rule Simplify-T is applied to simplify a member of G by another member of G, then the resulting simplified polynomial must be moved to S to await further processing. Thankfully, the theory of Abstract Gröbner Bases provides us with a simple method for proving the correctness and termination of the resulting algorithms which allows

Input: $\langle S = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\vec{x}], G = \emptyset, \prec \rangle$
Output: G s.t. G is a GBasis of S w.r.t. $\prec$
while $S \neq \emptyset$ **do**
  Invariant: G and S are maximally simplified w.r.t. G
  Choose $p \in S$
  Apply Orient to p
  Let q be the resulting oriented polynomial (in G)
  Use q to simplify G as long as possible
    using Simplify-H and Simplify-T
  Use G to simplify S as long as possible
    using Simplify-S
  Let $S_{old} := S$
  Apply Superpose to all pairs $\langle r, q \rangle$ $(r \in G, r \neq q)$
    for which Superpose has not been previously
    applied
  Let $S_{new} := S \setminus S_{old}$
  Use G to simplify members of S in S_{new}
    as long as possible using Simplify-S
  Apply Delete if possible
  if $((G \cup S) \cap (\mathbb{Q} \setminus \{0\}) \neq \emptyset)$ **then**
    Set $G := \{1\}$
    Set $S := \emptyset$
  end if
end while

Figure 5.1: GB algorithm based on OTTER saturation loop

Input: $\langle S = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\vec{x}], G = \emptyset, \prec \rangle$
Output: G s.t. G is a GBasis of S w.r.t. $\prec$
while $S \neq \emptyset$ **do**
  Invariant: G is maximally simplified w.r.t. G
  Choose $p \in S$
  Use G to simplify p as long as possible
    using Simplify-S
  Let s be the resulting simplified polynomial (in S)
  if $s \neq 0$ **then**
    Apply Orient to s
    Let q be the resulting oriented polynomial (in G)
    Use q to simplify G as long as possible
      using Simplify-H and Simplify-T
    Apply Superpose to all pairs $\langle r, q \rangle$ $(r \in G, r \neq q)$
      for which Superpose has not been previously
      applied
    Apply Delete if possible
    if $((G \cup S) \cap (\mathbb{Q} \setminus \{0\}) \neq \emptyset)$ **then**
      Set $G := \{1\}$
      Set $S := \emptyset$
    end if
  else
    Apply Delete to s
  end if
end while

Figure 5.2: GB algorithm based on DISCOUNT saturation loop

us to abstract away from these dynamics. This will be seen in **Section 5.2.3**.

If one contrasts these algorithms with Buchberger's Algorithm, we see a fundamental difference: In Buchberger's Algorithm, once a reduced S-polynomial is placed in G, it is never removed nor modified[2]. For L3 systems, this is ineffective. The reason is simple: With such a large number of input polynomials, G grows far too quickly when it is not continually simplified w.r.t. itself, causing the number of S-polynomials which must be considered, as well as the total number of possible polynomial reduction paths when one is using members of G to simplify other polynomials, to grow in each (non-terminating) iteration. Thus, both OTTER-GB and DISCOUNT-GB maintain the invariant that G is maximally simplified w.r.t. itself. This has the effect of keeping G "lean," continually purging G of redundant information. OTTER-GB maintains also the additional invariant that S is always maximally simplified w.r.t. G.

Returning to Buchberger's Algorithm, we see that only one form of simplification is done in Buchberger's Algorithm. This is the simplification of computed S-polynomials by members of the growing Gröbner basis. In Abstract GBs, we can characterise this type of simplification as that which arises when using members of G to simplify members of S. Only one rule lets us do this: Simplify-S. Borrowing the terminology from automated theorem proving (ATP), we call this *forward simplificaton*.

Both of our algorithms also perform another form of simplification: They use members of G to simplify other members of G. This is done through two rules, Simplify-H and Simplify-T. Note that the Simplify-H rule can be used to simplify a member of G by another member of G with the resulting simplified polynomial being allowed to stay in G. With Simplify-T, one uses a member of G to simplify another member of G, but the resulting simplified polynomial must be placed in S. Borrowing again ATP terminology, we call this *backward simplification*.

We can now state succinctly the key difference between Buchberger's Algorithm and those presented here: Buchberger's Algorithm only performs forward simplification. Both OTTER-GB and DISCOUNT-GB perform forward and backward simplification, but do so in different ways.

Finally, let us say a very short bit about another Gröbner basis algorithm, Faugére's F4, which we shall empirically compare with our algorithms on some L3 systems in **Section 5.3**. While also only performing forward simplification, Faugére's F4 differs

[2]Though one might, after the algorithm has finished and computed a Gröbner basis, proceed to minimise the computed Gröbner basis to a *reduced* Gröbner basis. Note, though, that during the initial Gröbner basis construction, once a polynomial is placed in G, it is then never modified. That is, this minimisation is done post facto w.r.t. Buchberger's Algorithm.

from Buchberger's Algorithm in that it (using deep insights from linear algebra) simplifies many S-polynomials simultaneously. This gives it much better performance than Buchberger's Algorithm on many classes of highly nonlinear problems, provided they have a relatively small total number of input polynomials. But, as backward simplification is never used, it does not cope much better than Buchberger's Algorithm when it comes to L3 systems. The situation is similar with the algorithm F5, as we will see.

In a primitive sense, one phenomenon related to the way in which F4 processes S-polynomials happens in the algorithms we present. In both OTTER-GB and DISCOUNT-GB, once a chosen polynomial is used to compute a set of S-polynomials against the other members of G, any of the deduced S-polynomials (in S) may become an immediate target for simplification, and the simplification steps of multiple S-polynomials may be interleaved. Crucially, the theory of Abstract GBs allows us to not care, at the level of correctness and termination proofs, about the order in which such reductions are performed. It would be very interesting to try and import some of the linear algebra underlying F4 into algorithms which still fit the abstract description of OTTER-GB and DISCOUNT-GB.

5.2.2 Term Indexing

As with terms in high-performance automated theorem proving [SRV01], it is imperative to have efficient methods for computing sets of polynomials which match a given polynomial w.r.t. the Abstract GB inference rules. These techniques need to answer queries of the form "which polynomials in the set of polynomials X can be used to perform an inference using rule R with polynomial p?".

It is fair to say that without such indexing methods, our new procedures would likely not perform better than those which were previously available. The reason is simple: In L3 systems, the size of the set of all retained polynomials (in Abstract GBs, this is the set $S_i \cup G_i$ for any state (S_i, G_i)) can be so large that answering questions like "Given a polynomial $p \in S_i$, which polynomials in G_i are candidates for reducing p via Simplify-S?" naively without some nontrivial form of filtering can lead to disaster. Both the mixture of forward and backward simplification present in our OTTER-GB and DISCOUNT-GB loops and the term indexing presented below are crucial to the performance improvements our algorithms have over previously available GB algorithms when it comes to L3 nonlinear systems.

There are many choices one can make when constructing term indexing schemes. Below, we present only the choices which are now used in the actual implementation of the OTTER-GB and DISCOUNT-GB algorithms in the SMT solver Z3. These are the indexing methods used in the experimental evaluation we present (comparing OTTER-GB and DISCOUNT-GB to state-of-the-art Gröbner basis algorithms available in mainstream computer algebra systems) in **Section 5.3**. Moreover, we present them at a high level of abstraction, just describing the key ideas underlying them.

There is much room here for an exploration of alternative indexing techniques, a discussion of trade-offs between different indexing choices, and so on. We do not go into that level of detail, nor have we done any systematic comparison of our current indexing schemes with other alternatives. This would be a very interesting undertaking which we hope to do some day. Instead, we simply developed basic techniques which allowed us to solve the problems we were aiming to solve, and we now recapitulate these concrete choices so that others may have the chance to build upon them.

5.2.2.1 Indexing for Superpose

For the application of Superpose, the indexing technique is based on superfluous S-polynomial criteria, in particular those we examined in **Section 3.4**. Recall that such a criterion is a computationally efficient sufficient condition for recognising when a given S-polynomial would reduce to zero w.r.t. the Gröbner basis being constructed, and thus signifies that the S-polynomial in question can be ignored. Such a criteria is in a sense a subsumption check, as an S-polynomial reducing to zero implies that all reductions it induces are present in the rewrite system induced by the portion of the Gröbner basis already constructed. Hence it would not contribute to obtaining a confluent rewriting system and need not be considered.

What is the goal of indexing for Superpose applications? Consider the query "which polynomials in the set of polynomials X can be used to perform a Superpose inference with polynomial p?" Imagine that $Answer(X,p)$ is our computed answer to that query. Naively, one might make $Answer(X,p) = X$. But, many of the polynomials in X may, when Superpose'd with p, lead to superfluous S-polynomials. For Superpose, we want our answers to be computed so that $Answer(X,p)$ contains very few polynomials q s.t. $\mathsf{spol}(p,q)$ is superfluous, and we want these answers to be computed efficiently. Thus, we want to have our term indexing techniques to some degree to take into account the superfluous S-polynomial criteria so that they ensure, as much as is feasible, that Superpose will only be applied when the generated S-polynomial would

not be superfluous.

There is, however, a difficulty in using such criteria in non-standard Gröbner basis loops: classical criteria, such as Buchberger 1 and 2 (what we in this thesis call **Criteria 1** and **2**), were originally proved correct only w.r.t. a fixed basis construction strategy, e.g., using an inductive cut-point argument w.r.t. the classical Buchberger's Algorithm [Buc79]. Given that our algorithms exhibit much different behaviour than Buchberger's Algorithm, it becomes nontrivial to establish the admissibility of the classical superfluous S-polynomial criteria w.r.t. algorithms like OTTER-GB and DISCOUNT-GB. Thankfully, we have solved this problem in **Chapter 3** for the superfluous S-polynomial criteria we consider, by proving these criteria admissible in the context of Abstract GBs.

In our implementation, we currently make use of **Criteria 1** and **3** in **Chapter 3**. We have not yet found an effective way to apply the second criteria we proved correct w.r.t. Abstract GBs, **Criterion 2**, in the context of term indexing.

It follows from **Criterion 3** that polynomials with linear leading monomials need not participate in Superpose inferences, given that both algorithms presented implement eager SH-simplification. Thus, for the application of Superpose, we index polynomials by their leading monomials so that given a polynomial p, we may quickly return lists of other polynomials whose leading monomials are (i) not relatively prime to p, and (ii) nonlinear. The index is essentially tracking the occurrences of variables in leading monomials of polynomials in G which have the potential to mate with p to contribute non-superfluous S-polynomials.

5.2.2.2 Indexing for Forward Simplification

When indexing for forward simplification, the polynomials used to do the simplification are the ones indexed. (This will be contrasted with indexing for backward simplification, where the targets of simplification are the ones indexed.) This means that the index for forward simplification should answer a query of the form "which polynomials in G can be used to forward simplify a polynomial p (in S)?". Moreover, it is only the *head* monomials of each polynomial in G which need to be indexed. For the application of forward simplification (Simplify-S), the indexing technique is based on the following observation.

Observation 5.2.1. For an oriented polynomial $p_2 = \underline{m_2} + q_2 \in G$ to be used to simplify an unoriented $p_1 = c_1 m_1 m_2 + q \in S$ using Simplify-S, it follows that (i) $\mathsf{totaldeg}(m_2)$

$\leq$ totaldeg$(m_1 m_2)$, and (ii) every variable in m_2 must appear in $m_1 m_2$.

While the above observation may seem a triviality, in practice it implies that to find an oriented polynomial which may Simplify-S a target monomial, we may place oriented polynomials in an index which facilitates (i) only considering polynomials whose leading monomial's total degree does not surpass that of the target, and (ii) in doing so only one variable of each leading monomial need be indexed.

Of course, there are many choices one can make when constructing such an index. We use a very simple data structure.

To build the forward simplification index

$$\texttt{fw\_index}\colon \texttt{Var} \times \texttt{Nat} \to 2^{\mathbb{Q}[\vec{x}]}$$

we process each $p \in G$ as follows:

> Let `v` be some[3] variable in the leading monomial of p
> Let `n` be the power of `v` in this monomial in p
> Add p to `fw_index[v][n]`

Now, to find polynomials which can rewrite a monomial m, we may perform a restricted search (letting $LM(q)$ denotes the leading monomial of q w.r.t. a background term ordering $\prec$):

for each variable `v` in m **do**
  Let `deg` = totaldeg(m)
  for `n` in $[1 \ldots \texttt{deg}]$ **do**
    for each `q` $\in$ `fw_index[v][n]` **do**
      if $(LM(\texttt{q})|m)$ **then**
        Record that `q` can be used to forward simplify monomial m
      end if
    end for
  end for
end for

In practice for L3 systems, most polynomials have low degree. So, we add a threshold in our index, and do not distinguish in the index between exponents with values greater than the threshold.

5.2.2.3 Indexing for Backward Simplification

When indexing for backward simplification, the goal is to quickly find the targets for simplification. (Contrast this with forward simplification above, where the goal was to

[3] In practice, we choose the *smallest* variable w.r.t. $\prec$, but this is not an essential choice.

quickly find the polynomials which could be used to perform simplification upon given targets.) This means that the index for backward simplification should answer a query of the form "which polynomials in G can be simplified by the polynomial $\underline{cm_1} + q$?". The index used for backward simplification (Simplify-H and Simplify-T) is the most expensive of the three, as it requires we index every monomial in every polynomial in G.

We can refine the above index query to the following more explicit one, which is what we really need to be able to answer efficiently for backward simplification: "Given a polynomial $\underline{m_1} + q$, what are the polynomials in G that contain monomials of the form cm_1m_2?" (In fact, the query can be refined even further when one properly implements structure sharing, so that identical monomials appearing in multiple polynomials in G actually correspond to the same "monomial object" in memory. In this case, one needs to answer: "Given a polynomial in $\underline{m_1} + q$, what are the *pointers to monomials* of the form cm_1m_2." Though structure sharing methods are used under-the-hood in de Moura's Z3 implementation of OTTER-GB and DISCOUNT-GB, we do not in this thesis present the ideas underlying the index at this low, but practically useful, level of abstraction.

As with forward simplification, the index used for backward simplification will map a pair (v,n) consisting of a variable and a degree (natural number) to a set of polynomials:

$$\texttt{bw\_index:}\ \ \texttt{Var} \times \texttt{Nat} \mapsto 2^{\mathbb{Q}[\vec{x}]}.$$

This `bw_index` will map a pair (v,n) to a set of polynomials $X = \texttt{bw\_index}(v,n)$ s.t. X contains precisely the polynomials in G which contain a monomial m s.t. m contains v and the total multivariate degree of m, $deg(m)$, is at least n.

The main idea underlying how we query this index is based on the following simple observation: For the polynomial $\underline{m_1} + q$ to simplify a monomial m in a polynomial in G, *every* variable in m_1 must appear in m, and $deg(m_1) \leq deg(m)$ must hold. The question then arises: Which variable v appearing in m_1 should be used in constructing the query $\texttt{bw\_index}(v, deg(m_1))$? When looking for targets of $\underline{m_1} + q$, we can do the following:

- Choose a variable v in m_1 that *minimises*

$$|\ \texttt{bw\_index}(v,\ deg(m_1))\ |$$

  w.r.t. all other variables in m_1. That is, v is selected so that there is no other

variable v' in m_1 s.t.

$$|\,\texttt{bw\_index}(v',\ deg(m_1))\,| < |\,\texttt{bw\_index}(v,\ deg(m_1))\,|.$$

This choosing of a variable v which minimises the number of candidate targets returned by the index can be done cheaply by keeping a counter, `num_occs(v,n)`, which keeps track of how many polynomials appear in the set $\texttt{bw\_index}(v,n)$ for each v appearing in any monomial in any polynomial in G, and for each n ranging from 1 to the maximal total multivariate degree among all monomials appearing in polynomials in G. We do precisely this. This counter is updated for each variable appearing in each monomial of a polynomial p whenever p is added to G (i.e., each time the Abstract GB rule Orient is invoked).

Finally, it is worth mentioning why our indexing techniques can be so much simpler than those considered in automated theorem proving (e.g., those based on substitution trees [Gra95]), yet still effective (cf. **Section 5.3**). This is because our terms are always shallow and ground. Importantly, we never have to deal with unification, nor with terms which are anything except monomials or polynomials in sparse sum-of-monomials normal form.

5.2.3 Algorithm Correctness

Let us now use the theory of Abstract GBs to prove the correctness of OTTER-GB and DISCOUNT-GB.

Theorem 5.2.2. *OTTER-GB and DISCOUNT-GB are terminating, functionally correct GB algorithms.*

Proof. By admissibility of the superfluous S-polynomial criteria, term indexing may be ignored. By the definition of polynomial ideal, if $(G \cup S) \cap (\mathbb{Q} \setminus \{0\}) \neq \emptyset$ then $\mathcal{I}(G \cup S) = \mathbb{Q}[\vec{x}]$. Hence $\{1\}$ is a Gröbner basis for $\mathcal{I}(G \cup S)$ w.r.t. any $\prec$ and the setting of G to $\{1\}$ (and subsequent termination of the loop through the setting of S to $\emptyset$) leads to correct behaviour. By **Theorem 3.2.8**, correctness is guaranteed if the procedures are *fair*, implement eager S-simplification, and Superpose is applied at most once between any two polynomials. The latter two properties are obvious. To observe fairness, we show $\mathsf{SP}(\bigcup_{i\geq 1}\bigcap_{j\geq i} G_j) \subseteq \bigcup_{i\geq 1} S_i$. Suppose not. Then, there must be some persistent pair $p_1, p_2 \in \bigcup_{i\geq 1}\bigcap_{j\geq i} G_j$ s.t. $\mathsf{spol}(p_1,p_2) \notin \bigcup_{i\geq 1} S_i$. WLOG, assume $p_1 \in \bigcap_{j\geq k_1} G_j$ and $p_2 \in \bigcap_{j\geq k_2} G_j$ s.t. $(k_2 > k_1)$ and let k_1, k_2 be the least indices

with this property. Then Simplify-H and Simplify-T must not have been used to simplify p_1, p_2 beyond states k_1, k_2 resp., as this would violate persistence. Consider the pass of either loop in which the Orient step corresponding to state k_2 occurs. Since p_1 is also persistent in G beyond state k_2, it follows that in such a pass Superpose must have been applied between p_1 and p_2, or Superpose must have been skipped because it had been previously applied to p_1, p_2. In either case we have $\mathsf{spol}(p_1, p_2) \in \bigcup_{i \geq 1} S_i$. □

5.3 Experimental Results

To evaluate our implementation, we created sets of random benchmarks with 4 kinds of polynomials: (a) identity polynomials of form $x - y$, (b) difference polynomials of form $x - y + k$ where k is an integer constant, (c) general linear polynomials, and (d) general polynomials. We experimented with two distributions of these kinds, a *mostly-lin* distribution with 40%, 50%, 5%, 5% of the four kinds, which reflects distributions of L3 problems we see in practice, and a *non-lin* distribution with 100% of kind (d).

Our results are summarised in **Table 5.1**. Each row but the last shows the results for 10 benchmarks with #Polys polynomials in #Vars variables. The last row shows results with 4 hard algebro-geometric benchmark problems which have been used to demonstrate the value of the F4 algorithm. The *discount* and *otter* columns are for our two implementations, and the other three are for Gröbner basis algorithms available in Maple 13: the *m-fgb* column is for a compiled implementation of the F4 algorithm written by J.C. Faugère, the *m-f4* column is for a Maple re-implementation of F4, and the *m-buchb* column for the traditional Maple implementation of Buchberger's Algorithm. Each of the entries in a column has 3 components: the number in the *#i* sub-column is the number of problems where the computed Gröbner basis is $\{1\}$, i.e. the set of polynomial equations is inconsistent, the 2nd number in the *avtm* sub-column is the average run-time in seconds for those problems on which tests halted in under 10 seconds, and the number in parentheses is the number of tests which halted in under 10 seconds. If the runs of all problems were over 10 seconds, the 2nd and 3rd numbers are replaced by TO for *time-out*.

As expected, as the ratio of polynomials to variables increases, we get more constrained systems and more definitely inconsistent problems. With the mostly-linear problems, as problem size increases, we see our procedures perform significantly better than those in Maple. One reason for this is that algorithms based on the principles of Buchberger's Algorithm, which includes F4, classically have a quadratic prepro-

			discount		otter		m-fgb		m-f4		m-buchb	
Distrib	#Polys	#Vars	#i	avtm	#i	avtm	#i	avtm	#i	avtm	#i	avtm
mostly-lin	100	100	10	.00(10)	10	.00(10)	10	.05(10)	10	.14(10)	10	.45(10)
	100	200	3	.00(10)	3	.00(10)	3	.12(10)	3	.53(10)	3	1.30(10)
	100	500	1	.00(10)	1	.00(10)	1	.14(10)	1	.25(10)	1	1.27(10)
	1000	1000	10	.00(10)	10	.00(10)	0	TO	9	4.39(9)	0	6.96(7)
	1000	2000	4	.00(4)	4	.00(4)	0	TO	3	6.65(3)	0	TO
	1000	5000	0	.01(10)	0	.01(10)	0	TO	0	TO	0	TO
	10000	10000	10	.06(10)	10	.07(10)	0	TO	0	TO	0	TO
	10000	20000	10	.07(10)	10	.09(10)	0	TO	0	TO	0	TO
	10000	50000	4	.11(10)	4	.13(10)	0	TO	0	TO	0	TO
non-lin	10	5	8	.71(8)	9	.46(9)	10	.01(10)	10	.05(10)	10	.05(10)
	10	10	0	8.06(1)	0	2.08(2)	0	1.90(6)	0	1.74(4)	0	1.93(3)
	50	20	9	.61(9)	9	.34(9)	10	.05(10)	9	.24(9)	10	1.31(10)
	50	50	2	.00(2)	2	.00(2)	1	.03(1)	2	.18(2)	2	.27(2)
hard-a-g	5-8	5-8	0	TO	0	TO	0	.25(4)	0	3.91(3)	0	2.72(2)

Table 5.1: Experimental Results

cessing step of computing all initial pairs of non-identical polynomials of the input basis, whereas our algorithms do not require this for correctness.

We see with the general non-linear random problems the Maple algorithms are usually better, and, with the hard algebro-geometric problems, the Maple algorithms are far superior.

5.4 Future Work

We see one immediate way in which these algorithms may be improved. This involves the mapping of **Criterion 2** in **Chapter 3** into an appropriate modification of our term indexing routines for Superpose. While this would likely not drastically affect performance on L3 systems, it seems plausible that this would be a boon in practice for computing with large systems of polynomials which have a higher nonlinear component than those currently amenable to our methods.

5.5 Conclusion

We have leveraged work within the automated theorem proving community to aid the extension of core computer algebra techniques to a challenging new type of problem. In particular, we have designed, implemented, and evaluated new Gröbner basis construction algorithms based on a combination of the OTTER and DISCOUNT saturation loops and term indexing techniques derived from both high-performance theorem proving and superfluous S-polynomial criteria in Gröbner basis theory. These procedures have been observed to significantly outperform previously available Gröbner basis algorithms for large, largely linear (L3) nonlinear systems. While proving these new algorithms correct was nontrivial, it became easy given the theory of Abstract Gröbner Bases. We see this work as an exciting cross-pollination between core techniques of the theorem proving and computer algebra communities, and look forward to furthering this natural symbiosis.

Chapter 6

Combined Decision Techniques for Fragments of RCF

6.1 Introduction

In this chapter, we begin investigating a practically-minded approach to deciding classes of high-dimensional (many-variable) sentences in the $\exists$ fragment of **RCF**. One aspect of our pragmatic focus will be an interest in developing *sound but incomplete* proof procedures which are effective for (and, in fact, can be *tailored to*) classes of problems arising in practical verification applications. We will work to build up a heterogeneous arsenal of $\exists$ **RCF** proof procedures, each with their own strengths and weaknesses, which may be easily combined with each other. The goal is to ease the building of specialised $\exists$**RCF** proof procedures which heuristically combine different techniques in a way which allows the combined procedure to outperform the individual methods when they are used in isolation, at least w.r.t. some specific problem classes of interest. This will culminate in a principled framework for building and applying such combined proof procedures realised in our tool **RAHD** in **Chapter 8**.

In addition, we will be especially interested in manners in which scalable sound but incomplete procedures can be used to enhance the practical efficacy of sound and *complete* methods such as cylindrical algebraic decomposition (CAD) by, for instance, recognising when certain expensive computations can be avoided. This goal will be realised in **Chapter 7** when we present the framework of Abstract Partial CAD. All of the techniques we describe have been implemented in our tool **RAHD** which will be presented in **Chapter 8**. In what follows, we shall freely quote text written in the

introduction (**Chapter 1**) when we believe it will help the reader.

§

In attempting to make real algebraic decision methods scale to high-dimensional settings, we are faced with what seems to be a rather insurmountable obstacle: the full first-order theory of **RCF** has infeasible complexity, and there are currently no known *complete* methods for the $\exists$ fragment of **RCF** which seem to fare better in practice than full **RCF** quantifier elimination. As if this were not enough, the complexities of known **RCF** and $\exists$ **RCF** decision methods are dependent primarily upon the *dimension* of their input formulas. In this regard, scaling **RCF** decision methods to high-dimensional settings seems utterly hopeless. Let us recall the following results from **Chapter 1**:

Theorem 6.1.1 (Davenport-Heinz)**.** *There are families of n-dimensional **RCF** formulas of length $O(n)$ whose only quantifier-free equivalences must contain polynomials of degree $2^{2^{\Omega(n)}}$ and of length $2^{2^{\Omega(n)}}$.*

Theorem 6.1.2 (Grigor'ev-Vorobjov)**.** *The $\exists$ fragment of **RCF** can be solved in time singly exponential in dimension.*

Even the apparent good news found in this final result — that the $\exists$ fragment of **RCF** has an exponential speed-up over the full first-order theory — is misleading in a practical sense. Analysis by Hong [Hon91] suggests that known singly-exponential algorithms for $\exists$ **RCF** will perform much worse[1] than even *full first-order* quantifier elimination algorithms such as cylindrical algebraic decomposition for all but astronomically large input formulas. Yet, there is no denying the fact that applying a full quantifier elimination algorithm to decide the unsatisfiability (i.e., falsity over $\mathbb{R}$) of a formula such as

$$\exists x_1, \ldots, x_{100}(x_1 * x_1 + \ldots + x_{100} * x_{100} < 0)$$

is an obvious misappropriation of computational (and temporal) resources. While an example such as this may seem contrived, consider the fact that when an **RCF** decision method is used in the context of formal verification efforts, it is often fed huge

[1] There is a very recent development showing promise in the other direction: Galen Huntington's beautiful 2008 Berkeley PhD thesis, "Towards an efficient decision procedure for the existential theory of the reals," has shown that Canny's singly exponential decision method for $\exists$ **RCF**, a procedure not considered by Hong in his analysis (Hong's analysis was in 1991, and Canny's method was first fully published in 1993 [Can93]), can in fact be implemented and made to solve a number of very small (bivariate, quadratic) examples. While a practical implementation of Canny's method is still a long way off, this work leaves one with a compelling optimism towards the possibility that, in contrast to Hong's conclusions in 1991, practically useful singly exponential decision procedures for $\exists$ **RCF** may eventually be realised.

collections of machine-generated formulas which may very well be (un)satisfiable for extremely simple reasons. In addition, problems arising from a particular application domain often share similar structure which traditional general methods will fail to exploit. We have observed these phenomena first-hand with many of the applications users have made of our **RAHD** tool. Thus, it seems advantageous to investigate algorithmic proof methods which attempt to make "easy decisions" quickly. And when such easy decisions fail, it will be desirable if the computations undertaken in attempting them could contribute to lessening the workload required of more heavy-weight analysis procedures which may be subsequently applied. Finally, if one knows in advance that a large collection of "similar" problems will be encountered, it would be desirable to provide mechanisms for *specialising* the approach of the proof procedure to exploit structural aspects of the formula class whenever possible. These concerns give rise to a particular *combined* approach to developing practical proof procedures for $\exists$ **RCF**. Let us discuss this in a bit more depth.

6.1.1 Our Approach

At the highest level, we would like proof procedures for $\exists$ **RCF** which

- scale to problems of realistic size (especially in many variables),
- are customisable for classes of problems with similar structure.

In working to accomplish this, we are faced with a rather wonderful difficulty: there are *many* different approaches to making **RCF** decisions, each with their own strengths and weaknesses. These include

- quantifier elimination by Muchnik sign matrices [Sch04, MO02],
- quantifier elimination by Cohen-Hörmander sign matrices [MH05],
- quantifier elimination by partial cylindrical algebraic decomposition [Bro04],
- quantifier elimination by virtual term substitution [Wei97],
- Positivstellensatz witness search by the Tiwari method [Tiw05a],
- Positivstellensatz witness search by semidefinite programming [Har07],
- interval constraint propagation and related methods [GB06, FHR$^+$07, Neu90, Rat06],

- connected component sampling by Basu-Pollock-Roy **PSPACE** methods [BPR06],
- techniques based on complex triangulation for zero-dimensional systems [CMXY09],
- Nelson-Oppen-like "distributivity-free" combinations of separate decision procedures for the additive and multiplicative fragments [AF06],
- and many others.

We wish to take advantage of this vast variety of powerful (semi-)decision methods. Our general programme then has been to do roughly as follows:

1. Study deeply, implement, and experiment with a number of different approaches to making $\exists$ **RCF** decisions.
2. Develop new variants of these decision methods by devising methods to effectively *combine* them in compelling ways. Such combinations are compelling, for instance, if with them it possible to decide sentences outside of the practical reach of the individual decision methods when they are used in isolation.
3. Build a tool which incorporates the most compelling decision methods we have investigated thus far and provides a framework for developing, investigating and applying new combinational methods.

This programme has resulted both in a number of novel combined decision methods and in a principled approach (based upon a *proof strategy language*) for facilitating the arbitrary combination of a heterogeneous collection of **RCF** decision techniques in a working tool.

6.1.2 Our Contribution

The main contribution of this chapter is to show how a heterogeneous collection of real algebraic proof procedures may be compellingly combined in novel ways. These novel combinations will be most interestingly illustrated in the context of our framework of *Abstract Partial Cylindrical Algebraic Decomposition* (AP-CAD), which gives rise to a *family* of combined $\exists$ **RCF** proof procedures, each parameterised by a particular proof strategy (e.g., a sound but possibly incomplete $\exists$ **RCF** (semi-)decision method) for short-circuiting expensive computations during the construction of CAD cell trees.

We will present the framework of AP-CAD in **Chapter 7**. The purpose of the present chapter is to present a large collection of $\exists$ **RCF** proof procedures which can be combined in compelling ways. All of the methods given in this chapter have been implemented in our **RAHD** tool, and a proof strategy language is provided in **RAHD** allowing users to synthesise their own combinations of the methods we present. By the end of this chapter and the next one on AP-CAD, we will have presented the mathematics behind the most challenging atomic proof procedures available in **RAHD**. This will free us up to focus mostly on a user-oriented tool description and experimental evaluation in **Chapter 8**.

6.2 Simple Exact Interval Constraint Propagation

Interval constraint propagation (ICP) is a powerful technique for reasoning about nonlinear algebraic constraints. As a decision method for deciding the satisfiability of polynomial constraints over the real numbers, it is *unsound*: ICP may fail to recognise the unsatisfiability of a polynomial constraint system and return a non-empty interval box for each variable when in fact no point satisfying the constraint system exists.

Nevertheless, as a reasoning mechanism which can contribute to real algebraic decisions, both on its own and in the context of other procedures, it is versatile and powerful. Moreover, if an ICP analysis deduces that a constraint system is *unsatisfiable*, then there is a soundness criterion which guarantees that this is indeed the case. This criterion guarantees in fact a stronger property: if a constraint system is satisfiable, then the intervals computed for each variable by ICP are guaranteed to contain a solution. One must simply take care to only use ICP in a sound way.

6.2.1 Related work

ICP methods have been used heavily in the context of nonlinear real arithmetic. For example, the SMT solver iSAT [FHR$^+$07] insightfully exploits an intimate relationship between ICP methods and DPLL-based SAT solving to interleave ICP and propositional reasoning to decide difficult classes of $\exists$ **RCF** sentences with complex boolean structure. From a different perspective, Ratschan has developed powerful methods for using ICP in the context of quantifier elimination for so-called *robust* formulas which often arise in physically-oriented hybrid systems [Rat06]. There are many other examples of applications of ICP to nonlinear constraint solving. It is an immensely rich and

very active research area filled with sophisticated techniques far beyond the scope of this thesis.

6.2.2 Intuition and Difficulties

In its simplest form, one begins an ICP analysis by assigning every variable v in an algebraic constraint system φ a compact interval with rational endpoints $B_v \subset \mathbb{R}$, and then proceeds to apply interval *contractors* or *narrowing operators* which use the atomic constraints to refine the interval boxes until a fixed point is reached. Such contraction is often applied in the context of a *branch-and-prune* loop, in which non-empty intervals are split into a finite covering and ICP is applied recursively upon the subintervals [HMK97] [BG06]. Two very attractive aspects of ICP are

- by over-approximating interval boxes during contraction steps, fast machine floating-point arithmetic may be used, and
- interval contractors have been developed which allow one to reason about algebraic constraints involving transcendental and other special functions.

One shortcoming of ICP methods is that sign determinations are often inconclusive. Particular difficulty is encountered when terms are expanded *sums of squares*. For example, given the constraint

$$x_1 \geq x_2^2 - 2x_2 + 1,$$

classical ICP methods will fail to deduce $(x_1 \geq 0)$. This is because ICP methods do not in general fully take into account correlations between the values of expressions, even at times correlations between the same variable symbol. Things get very difficult when expressions bound within the same interval appear multiple times in complicated expressions; this gives rise to the deep "dependency problem" in interval analysis [Krä06]. In addition, computed interval boxes are often highly dependent upon the representation of a polynomial: using Horner versus sum-of-monomials normal form can make a significant difference in the accuracy of computed boxes [CG02]. Similarly, contraction upon a constraint whose polynomials are fully factored into irreducibles may result in radically different boxes than contraction upon the same constraint with the polynomial in a different form. That is to say, ICP techniques must often be augmented with other reasoning mechanisms to be effective in practice.

One way we attempt to alleviate these shortcomings in our tool **RAHD** is by combining ICP techniques with many other **RCF** inference mechanisms. For instance, one simple mechanism in **RAHD** will recognise if a polynomial is in fact a sum of square monomials, and in that case will derive an additional constraint stating that the polynomial is non-negative. Such derived facts can then help the ICP procedure constrain terms within tighter intervals down the line. When reading this section, it is good to keep in mind that the ICP techniques presented will be later combined with many other **RCF** techniques in this way.

Let us now proceed to present an exact ICP calculus for generalised intervals with rational and infinite endpoints and open and closed boundaries. This ICP calculus uses a standard underlying arithmetic of generalised intervals equivalent to that presented by Hickey, Ju, and van Enden in [HJVE01], with a restriction of the interval endpoint values to $\mathbb{Q} \cup \{\text{-}\infty, +\infty\}$. We will discuss popular ICP techniques which can be used with this generalised interval arithmetic when specialised types of problems arise (e.g., for those in which every variable has been bound within a compact interval). These techniques are powerful and well-known. In addition, we will present a simple interval contraction calculus designed to work for arbitrary $\exists$ **RCF** formulas in which variables, for instance, may have no explicit bounds given. Once we have described this ICP calculus, we will then proceed to exploit it in the context of other real algebraic (semi-)decision procedures in the sections that follow.

6.2.3 An ICP Calculus for Generalised Intervals

We will present an ICP calculus for reasoning about real polynomial constraints with respect to *generalised* intervals with open or closed boundary types and rational or infinite endpoints. This is due to the fact that $\exists$ **RCF** sentences we wish to reason about will not be required to provide explicit compact interval bounds upon all of their variables.

As reasoning about generalised interval arithmetic operations is rather technically involved, requiring numerous case-splits depending upon

- the signs and finiteness of endpoints, and
- the open, closed, left-open-right-closed or left-closed-right-open boundary types

of the intervals involved, we have formally verified most of the interval machinery below within the **ACL2** theorem prover [KMM00]. This formal verification has been

done not only with respect to this interval calculus itself, but with respect to the actual implementation of it within our tool **RAHD**. When we state a result which has been formally verified in this way, we will reference the name the theorem has been given within our accompanying **ACL2** proof script. Instructions on how to obtain and replay this proof script may be found in **Appendix A**.

We begin by defining arithmetic operations upon $\mathcal{E} = \mathbb{Q} \cup \{\text{-}\infty, +\infty\}$, which is the set of *interval endpoints* we will consider. These operations will be defined so that some particular applications of them may result in an $\perp$, an error. It will be clear that in the context of the exposed ICP operations upon *intervals*, such errors will never arise.

$$\eta(\text{-}\infty) = +\infty.$$

$$\eta(+\infty) = \text{-}\infty.$$

$$e_1 +_{\mathcal{E}} e_2 = \begin{cases} e_1 + e_2 & \text{if } e_1, e_2 \in \mathbb{Q}, \\ e_2 & \text{if } e_1 \in \mathbb{Q} \wedge e_2 \in \{\text{-}\infty, +\infty\}, \\ e_1 & \text{if } e_2 \in \mathbb{Q} \wedge e_1 \in \{\text{-}\infty, +\infty\}, \\ e_1 & \text{if } e_1, e_2 \in \{\text{-}\infty, +\infty\} \wedge e_1 = e_2, \\ \perp & \text{otherwise.} \end{cases}$$

Observe that subtraction between non-identical infinities is allowed and defined in a rather counterintuitive fashion below. That this strange arithmetic is only used in a correct way will be clear from the definition of subtraction between *intervals*, given shortly.

$$e_1 -_{\mathcal{E}} e_2 = \begin{cases} e_1 - e_2 & \text{if } e_1, e_2 \in \mathbb{Q}, \\ \eta(e_2) & \text{if } e_1 \in \mathbb{Q} \wedge e_2 \in \{\text{-}\infty, +\infty\}, \\ e_1 & \text{if } e_1 \in \{\text{-}\infty, +\infty\} \wedge e_2 \in \mathbb{Q}, \\ \text{-}\infty & \text{if } e_1 = \text{-}\infty \wedge e_2 = +\infty, \\ +\infty & \text{if } e_1 = +\infty \wedge e_2 = \text{-}\infty, \\ \perp & \text{otherwise.} \end{cases}$$

$$e_1 *_{\mathcal{E}} e_2 = \begin{cases} e_1 * e_2 & \text{if } e_1, e_2 \in \mathbb{Q}, \\ \bot & \text{if } e_1 \in \{\text{-}\infty, +\infty\} \wedge e_2 = 0, \\ \bot & \text{if } e_2 \in \{\text{-}\infty, +\infty\} \wedge e_1 = 0, \\ e_1 & \text{if } e_1 \in \{\text{-}\infty, +\infty\} \wedge e_2 \in \mathbb{Q} \wedge e_2 > 0, \\ \eta(e_1) & \text{if } e_1 \in \{\text{-}\infty, +\infty\} \wedge e_2 \in \mathbb{Q} \wedge e_2 < 0, \\ e_2 & \text{if } e_2 \in \{\text{-}\infty, +\infty\} \wedge e_1 \in \mathbb{Q} \wedge e_1 > 0, \\ \eta(e_2) & \text{if } e_2 \in \{\text{-}\infty, +\infty\} \wedge e_1 \in \mathbb{Q} \wedge e_1 < 0, \\ +\infty & \text{if } e_1, e_2 \in \{\text{-}\infty, +\infty\} \wedge e_1 = e_2, \\ \text{-}\infty & \text{if } e_1, e_2 \in \{\text{-}\infty, +\infty\} \wedge e_1 \neq e_2. \end{cases}$$

We will also need some basic (in)equality relations upon $\mathcal{E}$.

$$e_1 \leq_{\mathcal{E}} e_2 = \begin{cases} \textbf{true} & \text{if } e_1, e_2 \in \mathbb{Q} \wedge e_1 \leq e_2, \\ \textbf{true} & \text{if } e_1 = \text{-}\infty, \\ \textbf{true} & \text{if } e_2 = +\infty, \\ \textbf{false} & \text{otherwise.} \end{cases}$$

$$(e_1 <_{\mathcal{E}} e_2) = (e_1 \neq e_2 \wedge e_1 \leq_{\mathcal{E}} e_2).$$

$$(e_1 >_{\mathcal{E}} e_2) = (e_2 <_{\mathcal{E}} e_1).$$

$$(e_1 \geq_{\mathcal{E}} e_2) = (e_2 \leq_{\mathcal{E}} e_1).$$

$$(e_1 =_{\mathcal{E}} e_2) = (e_1 \leq_{\mathcal{E}} e_2 \wedge e_1 \geq_{\mathcal{E}} e_2) = (e_1 = e_2).$$

With arithmetic and simple relations upon interval endpoints now defined, let us first define our intervals as formal objects and then build an arithmetic upon them.

Definition 6.2.1 (Formal Interval)**.** Formally, an interval will be a 4-tuple

$$\langle b_l, l, r, b_r \rangle \in \{\text{`['}, \text{`]'}\} \times \mathcal{E} \times \mathcal{E} \times \{\text{`['}, \text{`]'}\}.$$

We will write $\mathbb{I}_F$ to denote the collection of *formal intervals*.

A formal interval corresponds to a connected component of the real line, through the notion of an *interval realiser*.

Definition 6.2.2 (Interval Realiser)**.** Given a formal interval $i = \langle b_l, l, r, b_r \rangle \in \mathbb{I}_F$, the *realisation of* i, $\mathcal{R}(i)$, is defined as follows:

$$\mathcal{R}(i) = \begin{cases} \{x \in \mathbb{R} \mid l \leq x \leq r\} & \text{if } l, r \in \mathbb{Q} \wedge b_l = \text{`['} \wedge b_r = \text{`]'}, \\ \{x \in \mathbb{R} \mid l < x \leq r\} & \text{if } l, r \in \mathbb{Q} \wedge b_l = \text{`]'} \wedge b_r = \text{`]'}, \\ \{x \in \mathbb{R} \mid l < x < r\} & \text{if } l, r \in \mathbb{Q} \wedge b_l = \text{`]'} \wedge b_r = \text{`['}, \\ \{x \in \mathbb{R} \mid l \leq x < r\} & \text{if } l, r \in \mathbb{Q} \wedge b_l = \text{`['} \wedge b_r = \text{`['}, \\ \{x \in \mathbb{R} \mid x < r\} & \text{if } l = \text{-}\infty \wedge r \in \mathbb{Q} \wedge b_r = \text{`['}, \\ \{x \in \mathbb{R} \mid x \leq r\} & \text{if } l = \text{-}\infty \wedge r \in \mathbb{Q} \wedge b_r = \text{`]'}, \\ \{x \in \mathbb{R} \mid l < x\} & \text{if } r = +\infty \wedge l \in \mathbb{Q} \wedge b_l = \text{`]'}, \\ \{x \in \mathbb{R} \mid l \leq x\} & \text{if } r = +\infty \wedge l \in \mathbb{Q} \wedge b_l = \text{`['}, \\ \mathbb{R} & \text{if } l = \text{-}\infty \wedge r = +\infty, \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that for convenience, we allow -∞ and +∞ to be endpoints of closed boundary types, with the exact same semantics as if their corresponding boundary types were open. For example,

$$\mathcal{R}(\langle \text{`['}, \text{-}\infty, +\infty, \text{`]'} \rangle) = \mathcal{R}(\langle \text{`]'}, \text{-}\infty, +\infty, \text{`['} \rangle) = \mathbb{R}.$$

A crucial use of our interval calculus will be to determine the *emptiness* of intervals. That is, given an interval $i \in \mathbb{I}_F$, we will want a simple check for deciding if $\mathcal{R}(i) = \emptyset$. If an interval for a term in a constraint system is determined to be empty in the context of ICP, then this will result in a judgment of *unsatisfiability* for the constraint system considered. We will use $\Theta(i)$ as the emptiness predicate for formal intervals.

$$\Theta(\langle b_l, l, r, b_r \rangle) = \begin{cases} \textbf{true} & \text{if } (b_l = \text{`['} \wedge b_r = \text{`]'}) \wedge ((l > r) \vee (l = +\infty) \vee (r = \text{-}\infty)), \\ \textbf{true} & \text{if } \neg(b_l = \text{`['} \wedge b_r = \text{`]'}) \wedge ((l \geq r) \vee (l = +\infty) \vee (r = \text{-}\infty)), \\ \textbf{false} & \text{otherwise.} \end{cases}$$

The following lemma relates realisability and formal emptiness.

Lemma 6.2.3 (Correctness of interval emptiness)**.**

$$\forall i \in \mathbb{I}_F \, (\mathcal{R}(i) = \emptyset \iff \Theta(i) = \textit{true}) \, .$$

Proof. **ACL2** proof script theorem name: `I-EMPTY-CORRECT`. □

Let us now build a simple arithmetic upon formal intervals. Correctness of these operations will be expressed in terms of interval realisation. For example, the correctness criterion for interval addition ($+_{\mathbb{I}_F}$) will be as follows:

$$\forall x, y \in \mathbb{R} \, \forall i_1, i_2 \in \mathbb{I}_F \, (x \in \mathcal{R}(i_1) \wedge y \in \mathcal{R}(i_2) \implies x + y \in \mathcal{R}(i_1 +_{\mathbb{I}_F} i_2)) \, .$$

We use Δ to denote a canonical empty formal interval such as $\langle \text{`[', } 1, 0, \text{`]'} \rangle$. We shall also want some simple selector functions to aid in computing the boundary types of intervals resulting from interval arithmetic operations.

$$\Xi_l(b) = \begin{cases} \text{`[' } & \text{if } b = \textbf{true}, \\ \text{`]' } & \text{otherwise.} \end{cases}$$

$$\Xi_r(b) = \begin{cases} \text{`]' } & \text{if } b = \textbf{true}, \\ \text{`[' } & \text{otherwise.} \end{cases}$$

In the arithmetical definitions that follow, let

$$i_1 = \langle b_{1,l}, l_1, r_1, b_{1,r} \rangle \quad \text{and} \quad i_2 = \langle b_{2,l}, l_2, r_2, b_{2,r} \rangle .$$

First, we define interval addition.

$$i_1 +_{\mathbb{I}_F} i_2 = \begin{cases} \Delta & \text{if } \Theta(i_1) \vee \Theta(i_2), \\ \langle b_l, l_1 +_{\mathcal{E}} l_2, r_1 +_{\mathcal{E}} r_2, b_r \rangle & \text{where } b_l = \Xi_l(b_{1,l} = \text{`[' } \wedge \, b_{2,l} = \text{`['}) \\ & \text{and } b_r = \Xi_r(b_{1,r} = \text{`]' } \wedge \, b_{2,r} = \text{`]'}). \end{cases}$$

Lemma 6.2.4 (Correctness of interval addition)**.**

$$\forall x, y \in \mathbb{R} \, \forall i_1, i_2 \in \mathbb{I}_F \, (x \in \mathcal{R}(i_1) \wedge y \in \mathcal{R}(i_2) \implies x + y \in \mathcal{R}(i_1 +_{\mathbb{I}_F} i_2)) \, .$$

Proof. **ACL2** proof script theorem name: `I-+-CORRECT`. □

Next, subtraction. Observe how this definition makes sense of the strange subtraction between opposite infinities in $\mathcal{E}$ which we defined via $(-_{\mathcal{E}})$ above.

$$i_1 -_{\mathbb{I}_F} i_2 = \begin{cases} \Delta & \text{if } \Theta(i_1) \vee \Theta(i_2), \\ \langle b_l, l, r, b_r \rangle & \text{where } b_l = \Xi_l(b_{1,l} = \text{`['} \wedge b_{2,r} = \text{`]'}) \\ & \text{and } b_r = \Xi_r(b_{1,r} = \text{`]'} \wedge b_{2,l} = \text{`['}), \\ & \text{and } l = \begin{cases} l_1 -_{\mathcal{E}} r_2 & \text{if } (l_1 -_{\mathcal{E}} r_2) \in \mathbb{Q}, \\ \text{-}\infty & \text{if } (l_1 -_{\mathcal{E}} r_2) \in \{\text{-}\infty, +\infty\} \end{cases} \\ & \text{and } r = \begin{cases} r_1 -_{\mathcal{E}} l_2 & \text{if } (r_1 -_{\mathcal{E}} l_2) \in \mathbb{Q}, \\ +\infty & \text{if } (r_1 -_{\mathcal{E}} l_2) \in \{\text{-}\infty, +\infty\}. \end{cases} \end{cases}$$

Lemma 6.2.5 (Correctness of interval subtraction)**.**

$$\forall x, y \in \mathbb{R}\, \forall i_1, i_2 \in \mathbb{I}_F\, (x \in \mathcal{R}(i_1) \wedge y \in \mathcal{R}(i_2) \implies x - y \in \mathcal{R}(i_1 -_{\mathbb{I}_F} i_2)).$$

Proof. **ACL2** proof script theorem name: `I---CORRECT`. □

Finally, we come to interval multiplication. This is the most challenging of the arithmetical operations we consider, and we will proceed by first defining a number of auxiliary operations which contribute to it. Before we do so, however, let us discuss intuitively what this operation must accomplish.

As with the previous interval arithmetic operations, there are two parts to the computation of the product of generalised intervals:

1. the computation of interval endpoints, which will take values in $\mathcal{E}$, and
2. the determination of interval boundary types, which will result in an interval which is either
   - closed ($\langle$`['$, -, -,$ `]'$\rangle$),
   - open ($\langle$`]'$, -, -,$ `['$\rangle$),
   - left-open-right-closed ($\langle$`]'$, -, -,$ `]'$\rangle$), or
   - left-closed-right-open ($\langle$`['$, -, -,$ `['$\rangle$).

For interval products, the computation of the interval endpoint values in $\mathcal{E}$ is straightforward when each endpoint is rational: Given $i_1 = \langle b_{1,l}, l_1, r_1, b_{1,r} \rangle$ and $i_2 = \langle b_{2,l}, l_2, r_2, b_{2,r} \rangle$ with $l_1, r_1, l_2, r_2 \in \mathbb{Q}$, the product $i_1 *_{\mathbb{I}_F} i_2$ should be of the form

$$\langle b_l, \mathbf{min}(l_1 * l_2, l_1 * r_2, r_1 * l_2, r_1 * r_2), \mathbf{max}(l_1 * l_2, l_1 * r_2, r_1 * l_2, r_1 * r_2), b_r \rangle.$$

Extending this to the case of arbitrary endpoints in $\mathcal{E}$ requires some care, as $(0 *_{\mathcal{E}} \pm\infty)$ is undefined and must be avoided. Following [HJVE01], we will introduce a classification of interval types below to handle this. Once the computation of the resulting interval endpoint values is complete, we will then present the machinery for determining the resulting boundary types b_l and b_r.

Definition 6.2.6 (Interval classification)**.** Given a non-empty formal interval $i = \langle b_l, l, r, b_r \rangle$ (i.e., $\mathcal{R}(i) \neq \emptyset$), the *interval classification* of i,

$$\mathfrak{C}(i) : \mathbb{I}_F \to \{M, Z, P, N\},$$

will be determined by the signs of the numbers contained in the closure of the realisation of i, $\mathcal{R}(\langle `[', l, r, `]' \rangle)$. The four possible values of $\mathfrak{C}(i)$ will stand for *Mixed*, *Zero*, *Positive* and *Negative*.

$$\mathfrak{C}(i) = \begin{cases} M & \text{if } (l <_{\mathcal{E}} 0 <_{\mathcal{E}} r), \\ Z & \text{if } (l =_{\mathcal{E}} 0 =_{\mathcal{E}} r), \\ P & \text{if } (0 \leq_{\mathcal{E}} l \leq_{\mathcal{E}} r) \wedge (r >_{\mathcal{E}} 0), \\ N & \text{if } (l \leq_{\mathcal{E}} r \leq_{\mathcal{E}} 0) \wedge (l <_{\mathcal{E}} 0). \end{cases}$$

Given this classification, we can determine the left and right endpoint values of the product of two non-empty generalised intervals

$$i_1 = \langle b_{1,l}, l_1, r_1, b_{1,r} \rangle \quad \text{and} \quad i_2 = \langle b_{2,l}, l_2, r_2, b_{2,r} \rangle$$

using the following two functions.

$$\mathcal{V}_l^{\Pi}(i_1,i_2) = \begin{cases} l_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = P \ \wedge \ \mathfrak{C}(i_2) = P, \\ r_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = P \ \wedge \ \mathfrak{C}(i_2) = M, \\ r_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = P \ \wedge \ \mathfrak{C}(i_2) = N, \\ l_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = M \ \wedge \ \mathfrak{C}(i_2) = P, \\ min(l_1 *_{\mathcal{E}} r_2, r_1 *_{\mathcal{E}} l_2) & \text{if } \mathfrak{C}(i_1) = M \ \wedge \ \mathfrak{C}(i_2) = M, \\ r_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = M \ \wedge \ \mathfrak{C}(i_2) = N, \\ l_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = N \ \wedge \ \mathfrak{C}(i_2) = P, \\ l_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = N \ \wedge \ \mathfrak{C}(i_2) = M, \\ r_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = N \ \wedge \ \mathfrak{C}(i_2) = N, \\ 0 & \text{if } \mathfrak{C}(i_1) = Z \ \vee \ \mathfrak{C}(i_2) = Z. \end{cases}$$

$$\mathcal{V}_r^{\Pi}(i_1,i_2) = \begin{cases} r_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = P \ \wedge \ \mathfrak{C}(i_2) = P, \\ r_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = P \ \wedge \ \mathfrak{C}(i_2) = M, \\ l_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = P \ \wedge \ \mathfrak{C}(i_2) = N, \\ r_1 *_{\mathcal{E}} r_2 & \text{if } \mathfrak{C}(i_1) = M \ \wedge \ \mathfrak{C}(i_2) = P, \\ max(l_1 *_{\mathcal{E}} l_2, r_1 *_{\mathcal{E}} r_2) & \text{if } \mathfrak{C}(i_1) = M \ \wedge \ \mathfrak{C}(i_2) = M, \\ l_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = M \ \wedge \ \mathfrak{C}(i_2) = N, \\ r_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = N \ \wedge \ \mathfrak{C}(i_2) = P, \\ l_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = N \ \wedge \ \mathfrak{C}(i_2) = M, \\ l_1 *_{\mathcal{E}} l_2 & \text{if } \mathfrak{C}(i_1) = N \ \wedge \ \mathfrak{C}(i_2) = N, \\ 0 & \text{if } \mathfrak{C}(i_1) = Z \ \vee \ \mathfrak{C}(i_2) = Z. \end{cases}$$

Lemma 6.2.7 (Correctness of product endpoint values)**.**

$$\forall x,y \in \mathbb{R} \, \forall i_1,i_2 \in \mathbb{I}_F \left(x \in \mathcal{R}(i_1) \ \wedge \ y \in \mathcal{R}(i_2) \implies x*y \in \mathcal{R}(\langle \text{`['}, \mathcal{V}_l^{\Pi}(i_1,i_2), \mathcal{V}_r^{\Pi}(i_1,i_2), \text{`]'} \rangle) \right).$$

Proof. This follows directly from **Theorem 6** of [HJVE01]. □

We now turn our attention to determining the boundary types of products of non-empty generalised intervals. In doing so, we will make use of the following boolean function

$$\Omega : \mathcal{E} \times \mathcal{E} \times \{\textbf{true}, \textbf{false}\} \times \{\textbf{true}, \textbf{false}\} \to \{\textbf{true}, \textbf{false}\}$$

which will be used to determine whether the product boundary types b_l and b_r should be closed (**true**) or open (**false**).

$$\Omega(v_l, v_r, \alpha, \gamma) = (\alpha \wedge \gamma) \ \vee \ (\alpha \wedge (v_l = 0)) \ \vee \ (\gamma \wedge (v_r = 0)).$$

It will be convenient to define the inverses of Ξ_l and Ξ_r.

$$\Upsilon_l(b) = \begin{cases} \textbf{true} & \text{if } b = \text{`[', } \\ \textbf{false} & \text{otherwise.} \end{cases}$$

$$\Upsilon_r(b) = \begin{cases} \textbf{true} & \text{if } b = \text{`]', } \\ \textbf{false} & \text{otherwise.} \end{cases}$$

Now, using the interval classification as above, we will define the following two functions for determining the left and right boundary types of a product of non-empty generalised intervals

$$i_1 = \langle b_{1,l}, l_1, r_1, b_{1,r} \rangle \quad \text{and} \quad i_2 = \langle b_{2,l}, l_2, r_2, b_{2,r} \rangle.$$

$$\mathcal{CL}_l^{\prod}(i_1,i_2) = \begin{cases} \Omega(l_1,l_2,\Upsilon_l(b_{1,l}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=P \wedge \mathfrak{C}(i_2)=P, \\ \Omega(r_1,l_2,\Upsilon_r(b_{1,r}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=P \wedge \mathfrak{C}(i_2)=M, \\ \Omega(r_1,l_2,\Upsilon_r(b_{1,r}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=P \wedge \mathfrak{C}(i_2)=N, \\ \Omega(l_1,r_2,\Upsilon_l(b_{1,l}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=P, \\ \Omega(l_1,r_2,\Upsilon_l(b_{1,l}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=M \\ & \quad \wedge (l_1 *_{\mathcal{E}} r_2 <_{\mathcal{E}} r_1 *_{\mathcal{E}} l_2), \\ \Omega(l_1,r_2,\Upsilon_l(b_{1,l}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=M \\ \quad \vee \Omega(r_1,l_2,\Upsilon_r(b_{1,r}),\Upsilon_l(b_{2,l})) & \quad \wedge (l_1 *_{\mathcal{E}} r_2 =_{\mathcal{E}} r_1 *_{\mathcal{E}} l_2), \\ \Omega(r_1,l_2,\Upsilon_r(b_{1,r}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=M \\ & \quad \wedge (l_1 *_{\mathcal{E}} r_2 >_{\mathcal{E}} r_1 *_{\mathcal{E}} l_2), \\ \Omega(r_1,l_2,\Upsilon_r(b_{1,r}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=N, \\ \Omega(l_1,r_2,\Upsilon_l(b_{1,l}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=N \wedge \mathfrak{C}(i_2)=P, \\ \Omega(l_1,r_2,\Upsilon_l(b_{1,l}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=N \wedge \mathfrak{C}(i_2)=M, \\ \Omega(r_1,r_2,\Upsilon_r(b_{1,r}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=N \wedge \mathfrak{C}(i_2)=N, \\ \textbf{true} & \text{if } \mathfrak{C}(i_1)=Z \vee \mathfrak{C}(i_2)=Z. \end{cases}$$

$$\mathcal{CL}_r^{\prod}(i_1,i_2) = \begin{cases} \Omega(r_1,r_2,\Upsilon_r(b_{1,r}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=P \wedge \mathfrak{C}(i_2)=P, \\ \Omega(r_1,r_2,\Upsilon_r(b_{1,r}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=P \wedge \mathfrak{C}(i_2)=M, \\ \Omega(l_1,r_2,\Upsilon_l(b_{1,l}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=P \wedge \mathfrak{C}(i_2)=N, \\ \Omega(r_1,r_2,\Upsilon_r(b_{1,r}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=P, \\ \Omega(r_1,r_2,\Upsilon_r(b_{1,r}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=M \\ & \quad \wedge (l_1 *_{\mathcal{E}} l_2 <_{\mathcal{E}} r_1 *_{\mathcal{E}} r_2), \\ \Omega(r_1,r_2,\Upsilon_r(b_{1,r}),\Upsilon_r(b_{2,r})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=M \\ \quad \vee \Omega(l_1,l_2,\Upsilon_l(b_{1,l}),\Upsilon_l(b_{2,l})) & \quad \wedge (l_1 *_{\mathcal{E}} l_2 =_{\mathcal{E}} r_1 *_{\mathcal{E}} r_2), \\ \Omega(l_1,l_2,\Upsilon_l(b_{1,l}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=M \\ & \quad \wedge (l_1 *_{\mathcal{E}} l_2 >_{\mathcal{E}} r_1 *_{\mathcal{E}} r_2), \\ \Omega(l_1,l_2,\Upsilon_l(b_{1,l}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=M \wedge \mathfrak{C}(i_2)=N, \\ \Omega(r_1,l_2,\Upsilon_r(b_{1,r}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=N \wedge \mathfrak{C}(i_2)=P, \\ \Omega(l_1,l_2,\Upsilon_l(b_{1,l}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=N \wedge \mathfrak{C}(i_2)=M, \\ \Omega(l_1,l_2,\Upsilon_l(b_{1,l}),\Upsilon_l(b_{2,l})) & \text{if } \mathfrak{C}(i_1)=N \wedge \mathfrak{C}(i_2)=N, \\ \textbf{true} & \text{if } \mathfrak{C}(i_1)=Z \vee \mathfrak{C}(i_2)=Z. \end{cases}$$

At last, we are ready to define interval multiplication.

$$i_1 *_{\mathbb{I}_F} i_2 = \begin{cases} \Delta & \text{if } \Theta(i_1) \vee \Theta(i_2), \\ \langle \Xi_l(\mathcal{CL}_l^{\prod}(i_1,i_2)), \mathcal{V}_l^{\prod}(i_1,i_2), \mathcal{V}_r^{\prod}(i_1,i_2), \Xi_r(\mathcal{CL}_r^{\prod}(i_1,i_2)) \rangle & \text{otherwise.} \end{cases}$$

Lemma 6.2.8 (Correctness of interval multiplication)**.**

$$\forall x,y \in \mathbb{R}\ \forall i_1,i_2 \in \mathbb{I}_F\,(x \in \mathcal{R}(i_1) \wedge y \in \mathcal{R}(i_2) \implies x*y \in \mathcal{R}(i_1 *_{\mathbb{I}_F} i_2))\,.$$

Proof. This follows directly from **Theorem 14** of [HJVE01]. □

Finally, when using the above interval arithmetic in the context of ICP, the set-theoretic intersection of two intervals will play an important role. As with interval multiplication, we define two functions which will be used to determine the closure status of the left and right boundary types of interval intersections.

$$\mathcal{CL}_l^{\cap}(i_1,i_2) = \begin{cases} \Upsilon_l(b_{1,l}) & \text{if } l_1 >_{\mathcal{E}} l_2, \\ \Upsilon_l(b_{2,l}) & \text{if } l_1 <_{\mathcal{E}} l_2, \\ \Upsilon_l(b_{1,l}) \wedge \Upsilon_l(b_{2,1}) & \text{otherwise.} \end{cases}$$

$$\mathcal{CL}_r^{\cap}(i_1,i_2) = \begin{cases} \Upsilon_r(b_{1,r}) & \text{if } r_1 <_{\mathcal{E}} r_2, \\ \Upsilon_r(b_{2,r}) & \text{if } r_1 >_{\mathcal{E}} r_2, \\ \Upsilon_r(b_{1,r}) \wedge \Upsilon_r(b_{2,r}) & \text{otherwise.} \end{cases}$$

Then, interval intersection is defined as follows.

$$i_1 \cap_{\mathbb{I}_F} i_2 = \begin{cases} \Delta & \text{if } \Theta(i_1) \vee \Theta(i_2), \\ \Delta & \text{if } (\neg(l_1 \leq_{\mathcal{E}} l_2 \wedge l_2 \leq_{\mathcal{E}} r_1) \wedge \neg(l_2 \leq_{\mathcal{E}} l_1 \wedge l_1 \leq_{\mathcal{E}} r_2)), \\ \langle \Xi_l(\mathcal{CL}_l^{\cap}(i_1,i_2)), & \\ \quad \mathbf{max}(l_1,l_2), & \\ \quad \mathbf{min}(r_1,r_2), & \text{otherwise.} \\ \quad \Xi_r(\mathcal{CL}_r^{\cap}(i_1,i_2))\rangle & \end{cases}$$

Lemma 6.2.9 (Correctness of interval intersection)**.**

$$\forall x \in \mathbb{R}\, \forall i_1, i_2 \in \mathbb{I}_F\, (x \in \mathcal{R}(i_1) \wedge x \in \mathcal{R}(i_2) \iff x \in \mathcal{R}(i_1 \cap_{\mathbb{I}_F} i_2)).$$

Proof. **ACL2** proof script theorem name: `I-INTERSECT-CORRECT`. □

6.2.3.1 Simple Generalised Interval Contraction

Armed with the above machinery for performing arithmetic upon generalised intervals, we now turn our focus to how such arithmetic can be applied in the context of making real algebraic decisions. The general approach we follow is known as *interval constraint propagation* (ICP). ICP combines real solution space analysis techniques based upon interval arithmetic (often referred to as *interval analysis*) with search-space exploration techniques adopted from constraint programming. From a very high level, given a conjunctive $\exists$ **RCF** formula φ, ICP proceeds as follows:

1. A collection of terms $t_1, \ldots, t_k$ (usually the variables appearing in φ) are associated with intervals $I_1, \ldots, I_k$ (usually compact) and the interval containment assumptions $t_i \in I_i$ are asserted.

2. The conjuncts of φ and the interval containment assumptions $t_i \in I_i$ are used by *interval contractors* (also known as *narrowing operators*) to refine the intervals I_i known to contain each t_i. This application of interval contractors to the conjuncts of φ may include the derivation of additional facts which help tighten intervals, e.g., if φ contains a zero-dimensional polynomial system, then the variables appearing in these equations may be solved for during narrowing, and so on.

ICP can be naturally seen as a search problem: Given φ as above, one is searching the space of interval contexts for a "minimal" — up to a specified threshold — association of terms (usually variables) of φ with containing intervals. Since the development of the influential **Newton** system [HMK97], much work on ICP has centered around a constraint processing loop known as *branch-and-prune*. This loop performs global search by dividing the search space into subregions and examining them recursively.

The branch-and-prune process is parameterised by two key pieces: a *consistency criterion* ("does the current region contain a point satisfying the input problem?") and a method for *sub-problem generation* ("how should the current region be divided into a finite covering of subregions?"). Typically, the consistency criterion is designed to be a feasibly computable check guaranteeing only a relaxed version of actual consistency: it may fail to recognise when a region does not contain a solution, but it should recognise common classes of inconsistencies very quickly. Common consistency criteria are *hull-consistency* and its further relaxation *box-consistency* [HMK97].

At a high level, when presented with these two parameters as well as a region of the search space S for a conjunctive formula φ and an interval width threshold w, a branch-and-prune loop operates as follows:

1. Threshold: If the "width" of the region S is smaller than w, then S is returned as a candidate region containing a solution.
   Otherwise, we continue below.

2. Pruning: The consistency criterion (say, C) is applied to see if S is C-consistent with the formula φ. If S is determined to be C-inconsistent with φ, then S is dropped as a candidate region containing a solution ($\emptyset$ is returned).

3. Branching: If S has been determined to be C-consistent with φ, then the method for sub-problem generation is applied to generate a covering of S by finitely many (pairwise disjoint) subregions. The overall loop is then called recursively upon each subregion, and the union of the results is returned.

There are many approaches to branch-and-prune based ICP, and a plethora of deep mathematical techniques have been developed for exploiting interval arithmetic in the context of pruning. Many of these approaches, such as those based upon the *Interval Newton Method*, work only for systems of equations [Neu90]. Others, such as those found in the tool **RSolver**, work only for so-called "robust" quantified systems of inequalities in which every variable has been a priori associated with a containing compact interval [Rat06].

In a way, the situation for ICP methods is quite similar to the situation for **RCF** decision methods as a whole: There are myriad ICP techniques which have been proposed and work only for specific restricted classes of input problems. This is especially true for equational systems. Because interval methods have found much use in natural science applications where obtaining approximate solutions to systems of nonlinear equations is often sufficient, a vast array of deep interval techniques have been developed each of which only work for restricted classes of systems of equations [BS95, SCX03, SAG03, SVJ00].

In the same spirit as our work in combining different **RCF** decision methods, some ICP researchers have proposed a hybrid approach in which heuristic combinations of different ICP techniques are used based upon the structure of the input problem. For instance, the system **RealPaver** uses chiefly constraint satisfaction techniques [Gra01] but will combine them with variants of the Interval Newton Method when a square system of equations can be extracted [GB06].

It is beyond the scope of this dissertation to present even a glimpse of these many different approaches to ICP. We will give below, however, one very simple pruning mechanism for general $\exists$ **RCF** formulas which we use in our tool **RAHD** but have not seen before in the literature.

One difficulty with ICP for general $\exists$ **RCF** problems is that variables appearing in them are not required to be bound within compact intervals. This contrasts with much work in ICP motivated by physical science applications where such bounding is required for the pruning techniques used[2]. In typical physical science applications,

[2]Even in the few tools which do accept variables not bounded within a compact interval, this use is usually discouraged (and users are encouraged to remedy the situation by enhancing such input problems

such bounding requirements are usually sensible and do not hinder a scientist's practical modeling ability. But, when using ICP in the context of proving mathematical assertions about the real numbers which may be completely removed from physical applications, the situation is of course rather different.

For general $\exists$ **RCF** problems, we need to reason about variables, or more generally polynomials, which have been bound within any generalised real interval (]-∞,+∞[by default). This reasoning will make use of the notions of interval context and extension.

Definition 6.2.10 (Interval context)**.** Given a collection of polynomials $S = \{q_1, \ldots, q_m\} \subset \mathbb{Q}[\vec{x}]$, an *interval context* for S,

$$\mathbb{IC} : S \to \mathbb{I}_F$$

is a mapping of members of S into the set $\mathbb{I}_F$ of generalised formal intervals.

Definition 6.2.11 (Interval context refinement)**.** If $\mathbb{IC}_1$ is an interval context for S_1 and $\mathbb{IC}_2$ is an interval context for S_2, then $\mathbb{IC}_2$ is an *interval context refinement* of $\mathbb{IC}_1$ iff

$$S_1 \subseteq S_2 \quad \wedge \quad \forall p \in S_1 \left(\mathcal{R}(\mathbb{IC}_1(p)) \supseteq \mathcal{R}(\mathbb{IC}_2(p))\right).$$

Definition 6.2.12 (Interval extension)**.** An interval function $F : \mathbb{I}_F^k \to \mathbb{I}_F$ is an *interval extension* of a real function $f : \mathbb{R}^k \to \mathbb{R}$ iff

$$\forall I_1, \ldots, I_k \in \mathbb{I}_F \; \forall r_1, \ldots, r_k \in \mathbb{R} \left(\bigwedge_{i=1}^{k} r_i \in \mathcal{R}(I_i) \implies f(r_1, \ldots, r_k) \in \mathcal{R}(F(I_1, \ldots, I_k)) \right).$$

There are a few things to note about interval extensions. First, there are often many possible interval extensions for a given real function. Trivially, an $F : \mathbb{I}_F^k \to \mathbb{I}_F$ which returns the formal generalised interval $\langle$']', -∞, +∞, '['$\rangle$ for all inputs is an interval extension of every $f : \mathbb{R}^k \to \mathbb{R}$. Of course, interval extensions which compute tight containing intervals are desired, but for many operations (especially interval extensions of transcendental functions) one must make tradeoffs between the tightness of the extensions and the hardness of their computation. Second, the correctness lemmata we proved about our interval arithmetic operations in the previous section actually show these operations to be interval extensions of their real counterparts. That is, $+_{\mathbb{I}_F}$ is an interval extension of real $+$, $-_{\mathbb{I}_F}$ an interval extension of real $-$, and $*_{\mathbb{I}_F}$ an interval extension of real $*$.

with more constraints derived from physical considerations). The **RealPaver** manual, for instance, instructs: "Should we use infinities [as interval endpoints] or not? Yes, but only if no more information is known about the variables. For instance, given a variable that represents a distance between two points on Earth, the domain $[0, 5e4]$ is preferred to $\mathbb{R}$ since $5e4$ is an upper bound of Earth's circumference." [GB06]

Given interval extensions for basic arithmetic operations, we now turn to obtaining interval extensions for polynomials. That is, viewing a polynomial $p \in \mathbb{Q}[\vec{x}]$ as a function from $\mathbb{R}^n \to \mathbb{R}$, we seek an interval extension $P : \mathbb{I}_F^n \to \mathbb{I}_F$ for p. There are many ways to go about this. For instance, the **Newton** system provides three possible interval extensions for polynomials: the natural interval extension, the distributed interval extension, and the Taylor interval extension [HMK97]. Typically, in a branch-and-prune setting, the effectiveness of the consistency criteria applied during pruning depends greatly upon the interval extension used.

We will make use only of the so-called natural interval extension. This extension is by far the simplest and most intuitive.

Definition 6.2.13 (Natural interval extension)**.** Let $p \in \mathbb{Q}[\vec{x}]$ and let $\mathbb{IC}$ be an interval context for $S \subset \mathbb{Q}[\vec{x}]$ s.t. $\{x_1, \ldots, x_n\} \subset S$. Then, the *natural interval extension* of p w.r.t. $\mathbb{IC}$ is obtained by forming a new expression P from p and evaluating it as follows:

1. all rational constants q in p are replaced by the formal interval $\langle `[', q, q, `]' \rangle$,
2. all variables x_i in p are replaced by their containing intervals given by the context $\mathbb{IC}(x_i)$,
3. all arithmetic operations on $\mathbb{R}$ are replaced by their corresponding interval operations (e.g., $+ \mapsto +_{\mathbb{I}_F}$, $- \mapsto -_{\mathbb{I}_F}$, $* \mapsto *_{\mathbb{I}_F}$).

Of course, the actual value of this interval extension will be dependent upon the order of evaluation of the arithmetic interval expressions within P. As we will further refine this interval extension below (and say more about polynomial representations), let us gloss over this aspect for now.

Observe how the natural interval extension only takes into account the intervals variables have been associated with in an interval context. This is the definition usually found in the interval analysis literature, where having nontrivial bounding information on variables is customary. When working with generalised intervals and arbitrary $\exists$ **RCF** formulas, however, we have found it useful to extend the natural interval extension to take into account all information which is available in an interval context, including information known about terms which are not variables. This gives rise to what we call the *term natural interval extension*. When computing this extension, it is useful to have a generalisation of interval contexts in which the domain of the context is all polynomials in $\mathbb{Q}[\vec{x}]$.

Definition 6.2.14 (Total interval context)**.** Let $\mathbb{IC} : S \to \mathbb{I}_F$ be an interval context for $S \subset \mathbb{Q}[\vec{x}]$. Then, the *total interval context* of $\mathbb{IC}$,

$$\mathcal{T}(\mathbb{IC}) : \mathbb{Q}[\vec{x}] \to \mathbb{I}_F$$

is obtained from $\mathbb{IC}$ as follows:

$$\mathcal{T}(\mathbb{IC})(p) = \begin{cases} \mathbb{IC}(p) & \text{if } p \in S, \\ \langle \text{`]'}, \text{-}\infty, +\infty, \text{`['} \rangle & \text{otherwise.} \end{cases}$$

Thus far, we have treated polynomials abstractly without recourse to their actual machine representation. We believe this is an important choice for this exposition, as otherwise many definitions become needlessly complicated, and it is easy to see how to instantiate the above abstract machinery to specific polynomial representations. For computing the term natural interval extension, however, we will work recursively over the term structure of polynomials, and will thus need to make some representation assumptions. In the definition below, let us abuse notation and assume that polynomials in $\mathbb{Q}[\vec{x}]$ have been presented in a particular fully parenthesised form with each arithmetic operation $(+, -, *)$ binary. It will be clear how to adapt the definition to other polynomial representations.

Definition 6.2.15 (Term natural interval extension)**.**

$$\mathcal{N}(p, \mathbb{IC}) = \begin{cases} \langle \text{`['}, p, p, \text{`]'} \rangle & \text{if } p \in \mathbb{Q}, \\ \mathcal{T}(\mathbb{IC})(p) & \text{if } p \in \{x_1, \ldots, x_n\}, \\ \mathcal{T}(\mathbb{IC})(p) \cap_{\mathbb{I}_F} (\mathcal{N}(p_1, \mathbb{IC}) +_{\mathbb{I}_F} \mathcal{N}(p_2, \mathbb{IC})) & \text{if } p = (p_1 + p_2), \\ \mathcal{T}(\mathbb{IC})(p) \cap_{\mathbb{I}_F} (\mathcal{N}(p_1, \mathbb{IC}) -_{\mathbb{I}_F} \mathcal{N}(p_2, \mathbb{IC})) & \text{if } p = (p_1 - p_2), \\ \mathcal{T}(\mathbb{IC})(p) \cap_{\mathbb{I}_F} (\mathcal{N}(p_1, \mathbb{IC}) *_{\mathbb{I}_F} \mathcal{N}(p_2, \mathbb{IC})) & \text{if } p = (p_1 * p_2). \end{cases}$$

Finally, we are able to present the collection of generalised interval contraction rules we use in **Figure 6.1**. These rules use the conjuncts in an $\exists$ **RCF** formula to refine an interval context. Given an $\exists$ **RCF** formula φ, we will write $\mathbb{IC}_1 \vdash_\varphi \mathbb{IC}_2$ to mean that the interval context $\mathbb{IC}_1$ has been refined to the interval context $\mathbb{IC}_2$ using one of these rules applied to $\mathbb{IC}_1$ and a conjunct of φ. We use the notation $\mathbb{IC}(p) := I$ to mean the interval context refinement of $\mathbb{IC}$ which is obtained by changing $\mathbb{IC}(p)$ to I and leaving all other data of $\mathbb{IC}$ the same.

Recall the structure of a branch-and-prune loop as previously presented. We make use of these rules within such a loop, but bound their use in pruning by a maximum

$$\text{G-1}\quad \frac{\mathcal{N}(\mathbb{IC},p)=\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \qquad \mathcal{N}(\mathbb{IC},q)=\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \qquad (p>q)}{\mathbb{IC}(p):=\left(\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \cap_{\mathbb{I}_F} \langle \text{`]'},l_q,+\infty,\text{`['}\rangle\right)}$$

$$\text{G-2}\quad \frac{\mathcal{N}(\mathbb{IC},p)=\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \qquad \mathcal{N}(\mathbb{IC},q)=\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \qquad (p>q)}{\mathbb{IC}(q):=\left(\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \cap_{\mathbb{I}_F} \langle \text{`]'},-\infty,r_p,\text{`['}\rangle\right)}$$

$$\text{GE-1}\quad \frac{\mathcal{N}(\mathbb{IC},p)=\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \qquad \mathcal{N}(\mathbb{IC},q)=\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \qquad (p\geq q)}{\mathbb{IC}(p):=\left(\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \cap_{\mathbb{I}_F} \langle b_{1,q},l_q,+\infty,\text{`['}\rangle\right)}$$

$$\text{GE-2}\quad \frac{\mathcal{N}(\mathbb{IC},p)=\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \qquad \mathcal{N}(\mathbb{IC},q)=\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \qquad (p\geq q)}{\mathbb{IC}(q):=\left(\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \cap_{\mathbb{I}_F} \langle \text{`]'},-\infty,r_p,b_{2,p}\rangle\right)}$$

$$\text{E-1}\quad \frac{\mathcal{N}(\mathbb{IC},p)=\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \qquad \mathcal{N}(\mathbb{IC},q)=\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \qquad (p=q)}{\mathbb{IC}(p):=\left(\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \cap_{\mathbb{I}_F} \langle b_{1,q},l_q,r_q,b_{2,q}\rangle\right)}$$

$$\text{E-2}\quad \frac{\mathcal{N}(\mathbb{IC},p)=\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \qquad \mathcal{N}(\mathbb{IC},q)=\langle b_{1,q},l_q,r_q,b_{2,q}\rangle \qquad (p=q)}{\mathbb{IC}(q):=\left(\langle b_{1,p},l_p,r_p,b_{2,p}\rangle \cap_{\mathbb{I}_F} \langle b_{1,q},l_q,r_q,b_{2,q}\rangle\right)}$$

Figure 6.1: Generalised Interval Contraction Rules

number of rule applications per consistency check. We have found them to be most useful in the context of bounded fixed-point interval context computation, with our input problems expressed in the form of primitive ("three address code") constraints [FHR+07]. We also make use of more classical interval narrowing operators for the basic arithmetic operations, such as those of Cleary [Cle87]. We will discuss the use of this ICP procedure in more detail when we present our tool **RAHD** in **Chapter 8** .

6.3 Tiwari Positivstellensatz Method and Extensions

The Tiwari Positivstellensatz method [Tiw05a] is in reality a *family* of proof procedures for $\exists$ **RCF**. In practice, the method is aimed at proving $\forall$ **RCF** sentences by refuting their $\exists$ duals. The method is refutationally complete relative to an oracle for controlling the introduction of a particular class of new definitions and selecting a monomial order. These introduced definitions use slack variables to give names to terms which could appear in a type of certificate for the unsatisfiability of an $\exists$ **RCF** formula known as a *Positivstellensatz* witness. Given that this oracle task is com-

putable, a variant of the Tiwari procedure exists which is in principle guaranteed to prove that any unsatisfiable ∃ **RCF** formula is indeed unsatisfiable, though it may fail to recognise the satisfiability of satisfiable formulas[3]. In practice, we make use only of a terminating and refutationally incomplete variant of this method, where a proof search bound is employed (and can be user-specified), only a restricted class of new definitions are introduced, and ICP methods are used for recognising when a witness for unsatisfiability has been found. This gives rise to a particularly effective procedure which quickly recognises the unsatisfiability of many "simple" types of unsatisfiable ∃ **RCF** formulas, and usually gives up quickly on problems which are beyond its reach.

6.3.1 Overview

The Tiwari method is based upon a fundamental result in real algebraic geometry known as the *Positivstellensatz*[4]. As with Hilbert's Weak Nullstellensatz over $\mathbb{C}^n$, the Weak Positivstellensatz guarantees that a system unsatisfiable over $\mathbb{R}^n$ gives rise to a simple type of algebraic proof object certifying this unsatisfiability. Unlike the complex case, the Positivstellensatz must deal not only with equations but also with inequalities. This makes Positivstellensatz certificates more intricate than the ideal membership identities we are used to with Hilbert's Weak Nullstellensatz.

Theorem 6.3.1 (1). *[$\mathbb{Q}[\vec{x}]$ variant of Krivine-Stengle Weak Positivstellensatz]*

*The conjunctive ∃ **RCF** sentence*

$$\varphi = \exists \vec{x}\,(\bigwedge_{i=1}^{k_0} p_i = 0) \wedge (\bigwedge_{i=1}^{k_1} q_i \geq 0) \wedge (\bigwedge_{i=1}^{k_2} s_i \neq 0) \quad \textit{with} \quad p_i, q_i, s_i \in \mathbb{Q}[\vec{x}],\ k_0, k_1, k_2 \geq 1$$

[3] In fact, as predicted by Tiwari in [Tiw05a], effective bounds have been obtained on the size of Positivstellensatz witnesses, proving the existence of a variant of the Tiwari method which is a decision procedure (i.e., complete for both unsatisfiable and satisfiable input problems). Of course, this type of completeness proof ("run the refutationally-complete proof procedure until it has exhausted every possible Positivstellensatz witness up to the known bound; if it has found no proof of unsatisfiability, the system must be satisfiable") does not lend itself to efficient proof search. These bounds are so astronomical that we only make use of the Tiwari method as a procedure for proving unsatisfiability.

[4] As noted by Marshall [Mar08], while the Positivstellensatz has for some time been credited solely to Stengle [Ste73], it has recently been realised that many of the core ideas were present already in Krivine [Kri64] and subsequently rediscovered by Stengle independently. It is interesting to note that they arrived at their Positivstellensatzen in rather different ways: Krivine used **RCF** quantifier elimination while Stengle used Lang's Homomorphism Theorem.

is ***false*** *over* $\mathbb{R}$ *iff*

$$\begin{aligned}
&\exists \mathcal{P} \in Ideal(p_1, \ldots, p_{k_0}) \\
&\exists \mathcal{Q} \in Cone(q_1, \ldots, q_{k_1}) \\
&\exists \mathcal{S} \in Monoid(s_1, \ldots, s_{k_2}) \\
&\quad s.t. \quad \mathcal{P} + \mathcal{Q} + \mathcal{S}^2 = 0
\end{aligned}$$

where

$$\begin{aligned}
Ideal(a_1, \ldots, a_h) &= \left\{ \sum_{i=1}^{h} a_i b_i \mid b_i \in \mathbb{Q}[\vec{x}] \right\}, \\
Cone(a_1, \ldots, a_h) &= \left\{ r + \sum_{i=1}^{h} t_i u_i \mid r, t_i \in \sum (\mathbb{Q}[\vec{x}])^2,\ u_i \in\ Monoid(a_1, \ldots, a_h) \right\}, \\
Monoid(a_1, \ldots, a_h) &= \left\{ \prod_{i=1}^{h} (a_i)^j \mid j \in \mathbb{N} \right\}, \\
\sum (\mathbb{Q}[\vec{x}])^2 &= \left\{ \sum_{i=1}^{v} (p_i)^2 \mid p_i \in \mathbb{Q}[\vec{x}] \ \wedge\ v \in \mathbb{N} \right\}.
\end{aligned}$$

This theorem guarantees us that the unsatisfiability of an (in)equational real polynomial system can always be proven using a very special argumentative form, with the proof taking the shape of an algebraic identity. The computational interest in this theorem is twofold: First, as shown by Parrilo [Par03], the search for such proofs can be reduced to a sequence of convex optimisation problems. (Tiwari's method uses a different mechanism, based on Gröbner bases, to search for such proofs.) Second, because they ultimately take the form of algebraic identities, the unsatisfiability proofs guaranteed to exist by this (Weak) Positivstellensatz have a simple easily verifiable structure. Harrison has made use of this in his powerful `REAL_SOS` tactic in HOL-Light [Har07], as the proofs can be found by external optimisation tools, and then the resulting proof objects may be verified foundationally within HOL-Light without having to place any trust in the external optimisation tool. To gain familiarity with the Positivstellensatz, let us give a small example presented by Harrison in that `REAL_SOS` paper:

Example 6.3.2. Consider proving the universal half of the quadratic root criterion

$$\forall a\ \forall b\ \forall c\ \forall x\ \left(ax^2 + bx + c = 0 \implies b^2 - 4ac \geq 0 \right)$$

by showing the inconsistency of the formula

$$\exists a\ \exists b\ \exists c\ \exists x\ \left(ax^2 + bx + c = 0\ \wedge\ 4ac - b^2 > 0 \right).$$

Then, the following identity can be seen as a proof of the falsity of this $\exists$ sentence guaranteed to exist by the Positivstellensatz:

$$(4ac - b^2) + (2ax + b)^2 + (-4a)(ax^2 + bx + c) = 0.$$

With this identity in hand, it is trivial to see that the falsity of the $\exists$ sentence follows by simple inequality reasoning: $4ac - b^2$ is assumed positive, $(2ax+b)^2$ is non-negative since it is a square, and $(-4a)(ax^2+bx+c)$ is assumed to be 0 since ax^2+bx+c is assumed to be 0. But, then their sum should be positive, yet their sum is trivially seen to be 0 by polynomial arithmetic alone. Thus, the original $\exists$ statement cannot be true over $\mathbb{R}$.

When we are dealing purely with equational systems over $\mathbb{R}^n$, the Positivstellensatz above reduces to the so-called Real Nullstellensatz[5], which, as with the complex case of Hilbert's Weak Nullstellensatz, involves only ideal membership identities.

Theorem 6.3.3 (Real Nullstellensatz)**.** *Let S be the system of polynomial equations*

$$\begin{aligned} p_1 &= 0, \\ &\vdots \\ p_k &= 0, \end{aligned}$$

with $p_i \in \mathbb{Q}[\vec{x}]$. Then S is unsatisfiable over $\mathbb{R}^n$ iff there exists a polynomial $P \in \mathcal{I}(\{p_1,\ldots,p_k\})$ which is a sum of a sum of squares of polynomials in $\mathbb{Q}[\vec{x}]$ and the rational constant 1 *as follows:*

$$\exists P \in \mathcal{I}(\{p_1,\ldots,p_k\}) \text{ s.t. } P = \left(\left(\sum_{i=1}^{m} s_i^2\right) + 1\right) \quad \textit{for} \quad s_1,\ldots,s_m \in \mathbb{Q}[\vec{x}].$$

Equivalently, S is unsatisfiable over $\mathbb{R}^n$ iff there exists sequences of polynomials $q_1,\ldots,q_k$, $s_1,\ldots,s_m \in \mathbb{Q}[\vec{x}]$ s.t.

$$\left(\sum_{i=1}^{k} p_i(\vec{x}) q_i(\vec{x})\right) = \left(\left(\sum_{i=1}^{m} s_i^2\right) + 1\right).$$

The basic idea of the Tiwari procedure is to combine ideal saturation (derivation of S-polynomials), the introduction of new definitions (fresh variables which "name" special polynomials), and change of basis transformations to "push down" a special

[5]De Moura and myself (and an anonymous referee) have obtained some simple preliminary results on the structure of Real Nullstellensatz witnesses [PdM09b]. However, we have decided this work to be outside of the scope of this thesis.

class of Positivstellensatz witnesses so that they appear within a Gröbner basis. Given a conjunctive $\exists$ **RCF** sentence

$$\varphi = \exists \vec{x}(\bigwedge_{i=1}^{k_0} p_i = 0) \wedge (\bigwedge_{i=1}^{k_1} q_i > 0) \wedge (\bigwedge_{i=1}^{k_2} s_i \geq 0) \quad \text{with} \quad p_i, q_i, s_i \in \mathbb{Q}[\vec{x}],$$

this happens in four steps, though steps 2-4 may be interleaved:

1. Two classes of slack variables $(w_i), (v_i)$ are introduced to represent inequalities equationally by replacing $q_i > 0$ with $q_i - v_i = 0$, $s_i \geq 0$ by $s_i - w_i = 0$, and recording side conditions $v_i > 0$ and $w_i \geq 0$.

2. Given a term ordering, a Gröbner basis for the resulting system of equations is computed.

3. Members of the basis are checked for strict non-nullity by taking into account the slack variable side conditions.

4. If no non-null witness was found, new definitions are introduced and a new term ordering chosen so as to make a special type of Positivstellensatz witness smaller under some well-founded ordering. The process then repeats. In practice, we only perform the search upto a fixed number of iterations which may be user-specified.

In our particular variant, the non-nullity checking in step 3 is done by ICP (taking into account the slack variable side conditions) and occurs during the computation of the Gröbner basis at user specified intervals (after n new S-polynomials have been derived). This has the advantage that in practice, a full Gröbner basis often need not be computed before a witness is found.

This tight integration between Positivstellensatz search and ICP has another particularly nice byproduct: Consider $\mathbb{IC}$ as an interval context for an equational variant of φ as obtained using slack variables in step 1 above. Then, a run of the Tiwari procedure can also tighten known intervals $\mathbb{IC}(t)$ for polynomials t not involving slack variables. This is because during Gröbner basis normal form computation w.r.t. a Gröbner basis G, we have that $t_1 \mapsto_G t_2 \mapsto_G \ldots \mapsto_G t_k$ implies that containing intervals for $t_1, t_2, \ldots, t_k$ can always be soundly narrowed to $\bigcap_{i=1}^{k} \mathcal{R}(\mathbb{IC}(t_i))$. This holds for all polynomials $t_i \in \mathbb{Q}[\vec{x}, \vec{w}, \vec{v}]$ and can thus be used for $t_i \in \mathbb{Q}[\vec{x}]$. Thus, even if the Tiwari procedure fails in finding a witness to the unsatisfiability of φ, its integration with ICP may still contribute information about the solution space of φ which is then useful to other proof procedures which will take into account the interval context $\mathbb{IC}$.

6.3.2 Tiwari Positivstellensatz Calculus

6.3.2.1 Preliminaries

Similar to the calculus presented for Abstract Gröbner Bases in **Chapter 3**, the Tiwari rules will operate on objects which we call *states*. Each state S will either be the single value $\bot$ or will be a tuple containing four components — the sets of variables (x_i), (v_i), (w_i), and a set of basis polynomials in $\mathbb{Q}[\vec{x},\vec{v},\vec{w}]$ as follows:

$$S = \langle X, V, W, P\rangle \quad \text{where} \quad X = (x_i),\ V = (v_i),\ W = (w_i),\ P \subset \mathbb{Q}[\vec{x},\vec{v},\vec{w}].$$

In the initial state of a run of the Tiwari procedure, (x_i) will be the variables appearing in φ, (v_i) the slack variables assumed to be strictly positive, (w_i) the slack variables assumed to be non-negative, and P will be the polynomials in an equational representation of φ w.r.t. these slack variables. (Throughout a Tiwari procedure run, these sign assumptions will continue to hold on our sets of variables (v_i) and (w_i), as we will make precise below). If a run of the procedure computes the state $\bot$, then this run will constitute a proof of the unsatisfiability over $\mathbb{R}$ of an $\exists$ **RCF** formula corresponding to the initial state. Let us formalise this.

Definition 6.3.4 (State formula)**.** Given a state $S = \langle X, V, W, P\rangle$ with $V = (v_i)_{i=1}^{k_1}$, $W = (w_i)_{i=1}^{k_2}$ and $P = (p_i)_{i=1}^{k_0}$, we associate with it an $\exists$ **RCF** formula, the *state formula of* S, $\mathbb{F}(S)$, as follows:

$$\mathbb{F}(S) = \begin{cases} (0=1) & \text{if } S = \bot, \\ \exists\vec{x}\exists\vec{v}\exists\vec{w}(\bigwedge_{i=1}^{k_0} p_i = 0) \wedge (\bigwedge_{i=1}^{k_1} v_i > 0) \wedge (\bigwedge_{i=1}^{k_2} w_i \geq 0) & \text{otherwise.} \end{cases}$$

We call the subformula $(\bigwedge_{i=1}^{k_1} v_i > 0) \wedge (\bigwedge_{i=1}^{k_2} w_i \geq 0)$ the *sign assumptions of* S. We will also use $\mathbb{F}_{QF}(S)$ to mean the quantifier-free matrix of the state formula of S.

Now, we can conveniently discuss the satisfiability of a *state* by examining the truth of its corresponding state formula.

Definition 6.3.5 (State satisfiability)**.** Let S be a state. Then we say S is *satisfiable* iff

$$\langle \mathbb{R}, +, *, -, 0, 1, < \rangle \models \mathbb{F}(S).$$

If a state is unsatisfiable, then it will be possible to use the Tiwari calculus to construct a witness to this unsatisfiability. A key property of the calculus is that it will allow us to construct witnesses of a special *easily recognisable* form. These witnesses

are *essentially* Positivstellensatz witnesses (e.g., a proper Positivstellensatz witness may be extracted from them), but they differ syntactically from a classical Positivstellensatz witness in that they are able to take into account the sign assumptions of the state. We call such a witness a *state witness*.

A beautiful property of a state witness is that it can be recognised by simple *local* reasoning which examines only the *sign* of each monomial. This is a property not enjoyed by general Positivstellensatz (or even Real Nullstellensatz) witnesses. In the general case, simply reasoning about the sign of each monomial is insufficient as multiple monomials may need to be combined into a perfect square term. For instance, to recognise $p = x^2 - 2x + 2$ as a Real Nullstellensatz witness, one does not simply examine the sign of each monomial in isolation, but rather realises that the monomial $-2x$ is actually a part of the perfect square $(x-1)^2$ and thus $p = (x-1)^2 + 1$. Contrast this with the polynomial $q = x^2y^4 + 2z^2 + 1$. In this case, one may recognise that q is a Real Nullstellensatz witness simply by examining the sign of each monomial. State witnesses will be like q in this respect, but they will be more general in that the sign assumptions of the state containing them may play an essential role.

Definition 6.3.6 (State witness)**.** Let S be a state s.t. $S = \langle X, V, W, P\rangle$ with $V = (v_i)_{i=1}^{k_1}$ and $W = (w_i)_{i=1}^{k_2}$. Then, $p = (\sum_{i=1}^{u} c_i m_i) \in \mathbb{Q}[\vec{x}, \vec{v}, \vec{w}]$ ($p \neq 0$) is a *state witness* for the unsatisfiability of S iff

$$\mathcal{W}(p) \in \{\{StrictNeg\}, \{StrictPos\}, \{Neg, StrictNeg\}, \{Pos, StrictPos\}\}$$

where

$$\mathcal{W}(p) = \bigcup_{i=1}^{u} \{\mathcal{W}_0(c_i m_i)\}$$

and

$$\mathcal{W}_0(c_i m_i) = \begin{cases} StrictPos & \text{if } c_i > 0 \wedge \mathcal{W}_1(m_i) = \{StrictPos\}, \\ Pos & \text{if } c_i > 0 \wedge \mathcal{W}_1(m_i) \in \{\{Pos\}, \{Pos, StrictPos\}\}, \\ Neg & \text{if } c_i < 0 \wedge \mathcal{W}_1(m_i) \in \{\{Pos\}, \{Pos, StrictPos\}\}, \\ StrictNeg & \text{if } c_i < 0 \wedge \mathcal{W}_1(m_i) = \{StrictPos\}, \\ Unknown & \text{otherwise} \end{cases}$$

with $\mathcal{W}_1$ defined below assuming $(m_i = \prod_{j=1}^{t} z_j^{d(j)})$ s.t. each $z_j \in (X \cup V \cup W)$ as

$$\mathcal{W}_1(m_i) = \mathcal{W}_1(\prod_{j=1}^{t} z_j^{d(j)}) = \bigcup_{j=1}^{t} \left\{\mathcal{W}_2(z_j^{d(j)})\right\}$$

and

$$\mathcal{W}_2(z_j^{d(j)}) = \begin{cases} StrictPos & \text{if } z_j \in V, \\ Pos & \text{if } z_j \in W \ \vee \ d(j) \text{ is even}, \\ Unknown & \text{otherwise.} \end{cases}$$

Given any state $S = \langle X, V, W, P \rangle$, we externally maintain sets of variables V_{new}, W_{new} and X_{new} which are v_i's (resp. w_i's, x_i's) which do not yet appear in V (resp. W, X). When S is understood by the current context, we will simply write $v_i \in V_{new}$ to mean some "fresh" v_i which does not appear in V. These variables will be used to extend V, W and X with new variables used to "name" specially selected terms in polynomials appearing in P. For any state, $S = \langle X, V, W, P \rangle$, we will say "the sign assumptions of S" or "the state sign assumptions" to mean the collection of assumptions stating that every variable in V is strictly positive and every variable in W is non-negative.

A system equivalent to Tiwari's original (oracle-relative) refutationally complete calculus [Tiw05a] is presented in **Figure 6.2**. We have simplified the presentation slightly by assuming the Gröbner basis rule results in a basis consisting of monic polynomials w.r.t. $\prec$. The conditions on the Extend rules are rather notationally heavy and are not all straight-forward. Some, such as the Extend-2 and Extend-3 rules, are difficult to apply effectively in practice.

We will clarify the details so that the reader has a basic understanding of the generality and power of the original Tiwari method. Then, we will present an incomplete variant of it (using ICP) which corresponds to the procedure we have actually implemented and use in practice. Before examining the calculus in detail, however, let us follow closely an elucidating description given by Tiwari in [Tiw05a] which yields a good intuitive picture of what the inference rules will accomplish.

6.3.2.2 Further Intuition

Given an unsatisfiable state $S_i = \langle X_i, V_i, W_i, P_i \rangle$, the calculus is designed to make it possible to apply a sequence of inference rules to obtain a run $S_i \vdash S_{i+1} \vdash \ldots \vdash S_{i+k}$ s.t. a state witness for the unsatisfiability of state S_{i+k} appears in $GB_{\prec}(P_{i+k})$ for some monomial order $\prec$. To see why this is so, we will reason in a manner similar to how we did in **Chapter 3**: We will place a well-ordering $\sqsubset$ upon *state witnesses*, and show that if the *minimal witness in* $\mathcal{I}(P_i)$ is not in $GB_{\prec}(P_i)$, then we can apply an inference rule which will result in a state S_{i+1} s.t. either $S_{i+1} = \bot$ or the *minimal witness in* $\mathcal{I}(P_{i+1})$ is smaller w.r.t. $\sqsubset$ than the *minimal witness in* $\mathcal{I}(P_i)$. The well-foundedness of $\sqsubset$ then

guarantees that a witness can in principle be eventually found.

Consider an unsatisfiable state $S_i = \langle X_i, V_i, W_i, P_i \rangle$ in which no witness appears. The sets of variables V_i and W_i contribute to the knowledge we have about the signs of monomials occuring in polynomials in P_i. As in the definition of state witness, let us label a monomial *Pos* if we know it is non-negative, *StrictPos* if we know it is strictly positive, and *Unknown* if we know no constraints on its range. For instance, given the polynomial $x_1^2 - 2x_1w_1 + w_1^2 + v_1$, we know that

- x_1^2 and w_1^2 are both non-negative and label them *Pos*,
- v_1 is strictly positive and label it *StrictPos*,
- but we know no constraints on the range of $-2x_1w_1$ and label it *Unknown*.

We would like to obtain a witness in $GB_{\prec}(P_{i+k})$; a witness will contain no *Unknown* monomials. There are two ways we can eliminate *Unknown* monomials. First, we can recognise them to be cross-product terms in perfect square polynomials. For example, $-2x_1w_1$ can be recognised to be a part of the perfect square $(x_1 - w_1)^2$. The rule Extend-2 is designed to make such inferences possible by prescribing a method for systematically giving names to the bases of such perfect squares (e.g., $(x_1 - w_1)$) as they are needed. Second, the *Unknown* monomials can simply be eliminated by cancellation. For example, since ideals are closed under addition, if the polynomials $\underline{v_1^2} - w_1w_2 + 1$ and $\underline{w_3^2} + w_1w_2 + w_3 - 1$ are in $\mathcal{I}(P_i)$ then they sum to give $w_3^3 + v_1^2 + w_3$ which is a state witness present in $\mathcal{I}(P_i)$. The difficulty however is that Gröbner basis computation need not perform such a cancellation. In this example, w_1w_2 will not be eliminated since neither of the leading terms of either polynomial divide it. The rule Extend-1 works to fix this by giving a name to such monomials so that there is freedom to move them around within a monomial order (otherwise *admissibility* might preclude any monomial order from making them the head of any polynomial in which they appear) and they can then become exposed for possible cancellations.

6.3.2.3 The Tiwari Calculus

We shall now go through each rule in turn, introducing the needed notation as it is encountered. We say nothing about the GB rule except to clarify that "$\prec$ MO" means "$\prec$ is a monomial order."

GB	$\dfrac{X,V,W,P}{X,V,W,GB_{\prec}(P)}$	**if** $\prec$ MO over $\mathbb{Q}[\vec{x},\vec{v},\vec{w}]$
Extend-1A	$\dfrac{X,V,W,P=P'\cup\{\underline{m}+q\}}{X,V,W\cup\{w_i\},P\cup\{w_i-m\}}$	**if** $m\in([W\cup V]\setminus[V])$, $w_i\in W_{new}$
Extend-1B	$\dfrac{X,V,W,P=P'\cup\{\underline{m}+q\}}{X,V\cup\{v_i\},W,P\cup\{v_i-m\}}$	**if** $m\in[V]$, $v_i\in V_{new}$
Extend-2	$\dfrac{X,V,W,P}{X\cup\{x_i\},V,W,P\cup\{x_i-m_0-\alpha m_1\}}$	**if** $\langle m_0,m_1\rangle$ occurs in P, $\alpha\in\mathbb{Q}$, $x_i\in X_{new}$
Extend-3	$\dfrac{X,V,W,P=P'\cup\{\underline{m_0}+q\}}{X\cup\{x_i\},V,W,P\cup\{x_i-m_1\}}$	**if** $m_1^2m_2=m_0m_3$, $m_2\in[W\cup V]^{0,1}$, $x_i\in X_{new}$, $\lvert m_1\rvert>1$
Detect	$\dfrac{X,V,W,P=P'\cup\{\underline{m}+q\}}{X,V,W,P\cup\{m,q\}}$	**if** $\underline{m}+q$ is strictly non-null w.r.t. $V>0,W\geq 0$
Witness	$\dfrac{X,V,W,P'\cup\{\underline{m}\}}{\bot}$	**if** $\underline{m}\in[V]$

Figure 6.2: Refutationally complete Tiwari Calculus

First, Extend-1A. Given a set of variables A, $[A]$ is used to denote the multiplicative monoid generated by A. Then, this rule is used to introduce a fresh name for a monomial m which is recognised to be non-negative for a simple reason: m is the product of elements of $(W \cup V)$. We then give m a new name using a fresh variable $w_i \in W_{new}$. We make the further requirement that $m \in ([W \cup V] \setminus [V])$ because if $m \in [V]$, then the sign assumption corresponding to m can be strengthened, since $m \in [V]$ means m is strictly positive w.r.t. the current state sign assumptions. The next rule will handle this case.

Second, Extend-1B. This rule is the analogue of the previous one for the case when the monomial $m \in [V]$. In this case, m is strictly positive w.r.t. the current state sign assumptions, and so we name m by a fresh variable $v_i \in V_{new}$.

Third, Extend-2. This rule introduces a new name for a binomial of the form $m_0 + \alpha m_1$ in the hopes that a polynomial of the form $(m_0 + \alpha m_1 + q)^2$ will appear in a Positivstellensatz witness. Let us explain the terminology of the side conditions. We use $[V]^{0,1}$ to denote the collection of members of $[V]$ in which variables in power-products appear with degree at most 1. We say that a power-product m *occurs directly in* P if there is a polynomial in P which contains a monomial with power-product m. We say a power-product m *occurs in* P *with factor* $m_0 \in [V]^{0,1}$ if there exists $m_1 \in [V]$ s.t. $m_1 | m m_0$ and m_1 occurs directly in P. Then, we say a pair of power-products $\langle m_0, m_1 \rangle$ *occurs in* P if

- $m_0 m_1$ occurs in P with factor m_2, and
- either $m_0^2 m_2$ occurs in P with factor 1 or $\frac{m_0^3 m_2}{u}$ occurs in P with factor 1 for some $u \in (W \cup V)$, and
- either $m_1^2 m_2$ occurs in P with factor 1 or $\frac{m_1^3 m_2}{u}$ occurs in P with factor 1 for some $u \in (W \cup V)$.

In the application of this rule, α is a symbol denoting a rational in $\mathbb{Q}$ and must be (eventually) instantiated. A value for α might not be known at the time of the application of this rule, however, so in the original presentation of the Tiwari method [Tiw05a], it is proposed that one do the following: Apply the rule and delay the determination of α until later by

- extending the field one works over in all inference rules from $\mathbb{Q}$ to $\mathbb{Q}(\alpha)$ (or, more generally, from $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)$ to $\mathbb{Q}(\alpha_1, \ldots, \alpha_{k+1})$),
- determining a rational value for α by computing the zero of some nontrivial linear expression over $\mathbb{Q}(\alpha)$ which, through use of the simplification inherent

in Gröbner basis computation (rule GB), will eventually appear as a coefficient of a monomial occuring directly in P. After instantiating α with this value, the monomial whose coefficient was used to derive this value for α will then be eliminated from the polynomial in which it appears (as its coefficient will evaluate to 0).

We see immediately one difficulty in orchestrating the Tiwari method in its refutationally complete form: There are *many* choices one can make in completing an application of rule Extend-2, as after simplification, α may appear in many different nontrivial linear expressions (each a coefficient for some monomial occuring directly in P), and these different expressions may of course have different zeroes. Interested readers should consult the original reference to learn more about this, paying particular attention to the second example on page 11 and the use of the rule Extend-2 in the proof of **Theorem 3** on page 12. In our implementation presented shortly (as well as the variant implemented and made available by Tiwari [Tiw05b]), this rule is not used at the expense of refutational completeness.

Fourth, Extend-3. As with the previous rule, this rule is used to capture cross-product terms which appear in perfect square polynomials in a Real Nullstellensatz witness. We use $|m_0|$ to denote the total degree of the power-product m_0. The difference between this rule and Extend-2 is that we need not find m_1 nor a value for α when $|m_0| > 1$. As with Extend-2, we do not know of implementations which actually use this rule.

6.3.3 A Practical Variant with ICP and External Saturation

We now present the variant of the Tiwari method we use in practice. To be precise, this variant is not actually an *instance* of the Tiwari method (e.g., some strategy for sequencing the rules of the calculus given in **Figure 6.2**), but is rather a closely related method — also presented as an abstract calculus — with two key differences from the original:

1. ICP is used throughout for recognising polynomials which should be named by fresh variables in V and W as well as for detecting unsatisfiability.

2. A rule (ST or "Saturate and Tighten") is provided to allow external saturation procedures to contribute information they have deduced from a state and make it available to the ICP engine in the hopes of enhancing interval contraction.

While it is easy to imagine that utilising ICP in this way may be advantageous, perhaps something should be said to motivate the integration of external saturation methods. We will cover a number of such saturation techniques in the next section, but let us now pick one small piece of such a method and argue why its integration in the manner presented in **Figure 6.3** might make sense.

Example 6.3.7. *Imagine applying the Tiwari method to the formula*

$$ax^2+bx+c=0 \ \wedge \ b^2-4ac<0$$

which leads to the initial state

$$S=\langle\{a,b,c,x\},\{v_1\},\emptyset,\{ax^2+bx+c,-b^2+4ac-v_1\}\rangle$$

which after application of the GB *rule with* $\prec$ = Lex$(a\succ b\succ c\succ x\succ v_1)$ *yields the equisatisfiable state*

$$S'=\langle\{a,b,c,x\},\{v_1\},\emptyset,\{[\underline{ac}-1/4b^2-1/4v,\underline{ax^2}+bx+c,\underline{b^2x^2}+4bcx+4c^2+x^2v]\}\rangle.$$

This simple state, unsatisfiable over $\mathbb{R}^5$*, poses a difficulty for the method, as one needs to use a combination of the* Extend-2 *and* Extend-3 *rules to obtain a state witness of unsatisfiability. If instead, one had an external method for* discriminant saturation *(cf.* ***Section 6.4.1.1****) which applied the rule*

$$\frac{p\in\mathbb{Q}[\vec{x}] \qquad deg(p,x)=2 \qquad p=0}{discriminant(p,x_i)\geq 0}$$

then $b^2-4ac\geq 0$ *would be deduced directly from* $ax^2+bx+c=0$*. This fact, combined with the state formula* $\mathbb{F}(S')$*, would be enough for ICP to obtain the unsatisfiability.*

Of course, the above example is trivial in the sense that discriminant saturation is all that was needed to obtain the proof of unsatisfiability. But, we have found the flexibility of allowing external saturation procedures to contribute facts in this way to be invaluable. Such deduced information often allows the ICP procedure to substantially tighten intervals, and our implemented variant is designed to recognise a state to be unsatisfiable whenever its interval context contains an association between a polynomial and an empty interval. We will see in **Chapter 8** how this is all done in the context of our tool **RAHD**. The core idea is that such saturation procedures are simply given as functional parameters to our implemented variant of the Tiwari method. Thus, one is able to use one's own strategic judgment as to the nature and extent of the external saturation which should be performed during Tiwari-style proof search.

Init-1A
$$\frac{\mathbb{IC}, X, V, W = W' \cup \{w_i\}, P}{(\mathbb{IC}(w_i) := \mathcal{N}(\mathbb{IC}, w_i) \cap_{\mathbb{I}_F} \langle \text{`[`}, 0, +\infty, \text{`[`} \rangle), X, V, W, P}$$

Init-1B
$$\frac{\mathbb{IC}, X, V = V' \cup \{v_i\}, W, P}{(\mathbb{IC}(v_i) := \mathcal{N}(\mathbb{IC}, v_i) \cap_{\mathbb{I}_F} \langle \text{`]`}, 0, +\infty, \text{`[`} \rangle), X, V, W, P}$$

GB
$$\frac{\mathbb{IC}, X, V, W, P}{\mathbb{IC}, X, V, W, GB_{\prec}(P)}$$
if $\prec$ MO over $\mathbb{Q}[\vec{x}, \vec{v}, \vec{w}]$

Ext-A
$$\frac{\mathbb{IC}, X, V, W, P = P' \cup \{\underline{m} + q\}}{\mathbb{IC}, X, V, W \cup \{w_i\}, P \cup \{w_i - m\}}$$
if $m \in ([W \cup V] \setminus [V])$, $w_i \in W_{new}$

Ext-B
$$\frac{\mathbb{IC}, X, V, W, P = P' \cup \{\underline{m} + q\}}{\mathbb{IC}, X, V \cup \{v_i\}, W, P \cup \{v_i - m\}}$$
if $m \in [V]$, $v_i \in V_{new}$

Ext-ICP-A
$$\frac{\mathbb{IC}, X, V, W, P = P' \cup \{q_1 + q_2\}}{\mathbb{IC}, X, V, W \cup \{w_i\}, P \cup \{w_i - q_1\}}$$
if $\mathcal{N}(\mathbb{IC}, q_1) \cap_{\mathbb{I}_F} \langle \text{`]`}, \text{-}\infty, 0, \text{`[`} \rangle = \Delta$, $w_i \in W_{new}$

Ext-ICP-B
$$\frac{\mathbb{IC}, X, V, W, P = P' \cup \{q_1 + q_2\}}{\mathbb{IC}, X, V \cup \{v_i\}, W, P \cup \{v_i - q_1\}}$$
if $\mathcal{N}(\mathbb{IC}, q_1) \cap_{\mathbb{I}_F} \langle \text{`]`}, \text{-}\infty, 0, \text{`]`} \rangle = \Delta$, $v_i \in V_{new}$

ST
$$\frac{\mathbb{IC}, X, V, W, P}{\mathbb{IC}', X, V, W, P}$$
if $A = \mathfrak{S}(\mathbb{F}_{QF}(\langle X, V, W, P \rangle))$, $\mathbb{IC} \vdash_{ICP(A)} \mathbb{IC}'$

Tighten
$$\frac{\mathbb{IC}, X, V, W, P}{\mathbb{IC}', X, V, W, P}$$
if $\mathbb{IC} \vdash_{ICP(\mathbb{F}_{QF}(\langle X, V, W, P \rangle))} \mathbb{IC}'$

Unsat
$$\frac{\mathbb{IC}, X, V, W, P}{\bot}$$
if $\exists p \in \mathbb{Q}[\vec{x}, \vec{v}, \vec{w}]$ s.t. $\mathcal{N}(\mathbb{IC}, p) = \Delta$

Figure 6.3: ICP-based Variant of the Tiwari Calculus with External Saturation

Let us say a few things about the calculus given in **Figure 6.3**. Observe that the states now have one additional component: an interval context, usually written $\mathbb{IC}$, as developed in **Section 6.2**. It is trivial to adapt the relevant state formula and satisfiability definitions developed previously to this new setting.

As noted, interval contexts partake in deductions in this calculus in a number of ways. First, once variables with implicit sign assumptions (v_i, w_i) are introduced to name monomials using the Ext-A and Ext-B rules, the Init-1A and Init-1B rules can then be used to update the interval context with this implicit sign information.

Second, the rules Ext-ICP-A and Ext-ICP-B are used to give names to arbitrary terms appearing in polynomials in the state polynomial system. This is done when ICP upon the state formula has recognised a term appearing in a polynomial to be either non-negative or strictly positive. This is particularly nice using ICP, as to find such terms, one simply looks through the interval context of the state, flagging each polynomial whose associated interval has an empty intersection with either $\langle$'$]$', -∞, 0, '$[$'$\rangle$ or $\langle$'$]$', -∞, 0, '$]$'$\rangle$.

Third, we have the rule ST. Let us clarify the notation. We write $\mathbb{IC} \vdash_{ICP(S)} \mathbb{IC}'$ to mean that the interval context $\mathbb{IC}'$ was obtained from the interval context $\mathbb{IC}$ by some application of the ICP rules in **Figure 6.1** with atomic formulas in the state formula of S serving as the polynomial constraint hypotheses. Thus, it follows that $\mathbb{IC}'$ will be an *interval context refinement* of $\mathbb{IC}$ (cf **Definition 6.2.11**). The rule ST ("Saturate and Tighten") is used to allow an external saturation or deduction procedure to derive information from a state which is then used to enhance interval contraction upon the state's interval context. In this rule, $\mathfrak{G}$ is such a saturation procedure which maps a quantifier-free conjunctive formula F_1 over $\mathbb{Q}[\vec{x}, \vec{v}, \vec{w}]$ to another formula F_2 s.t. $\mathbf{RCF} \models F_1 \implies F_2$. In particular, $\mathfrak{G}(\mathbb{F}_{QF}(\langle X, V, W, P\rangle))$ is the result of applying such a saturation procedure to the quantifier-free matrix of the state formula.

Fourth, we have the rule Tighten. This rule simply allows one to invoke the ICP procedure at any time so as to further contract known intervals based upon the state formula. This is equivalent to ST with $\mathfrak{G}$ an identity function.

Finally, we have the rule Unsat. This rule allows one to determine the current state to be unsatisfiable when its interval context associates any polynomial with an empty interval.

Given the soundness of our ICP method and the allowed external saturation procedures, the following lemma is immediate.

Lemma 6.3.8 (Soundness of Extended Tiwari Method)**.** *Let*

$$\varphi = \exists \vec{x}(\bigwedge_{i=1}^{k_0} p_i = 0) \wedge (\bigwedge_{i=1}^{k_1} q_i > 0) \wedge (\bigwedge_{i=1}^{k_2} s_i \geq 0) \quad \textit{with} \quad p_i, q_i, s_i \in \mathbb{Q}[\vec{x}],$$

with

$$S = \langle \mathbb{IC}_\emptyset, X, V, W, P \rangle$$

s.t. $\langle X, V, W, P \rangle$ *is the initial state associated with* φ *in the original Tiwari method (cf* ***Section 6.3.2.1****) and* $\mathbb{IC}_\emptyset$ *is the empty interval context. Then,*

$$S \vdash \bot \quad \Longrightarrow \quad \textbf{\textit{RCF}} \models \neg\varphi,$$

where $S \vdash S'$ *means* S' *is obtainable from* S *using the rules in* ***Figure 6.3****.*

We will see how this method may be applied in the context of our tool **RAHD** in **Chapter 8**. Carrying on with the current chapter, we will now proceed to build up some simplification and saturation machinery, which will yield some concrete choices for the functional parameter $\mathfrak{G}$ in rule ST. Once this is done, we will examine how this hierarchy of combined procedures can be utilised within CAD-based decision methods by introducing the framework of *Abstract Partial Cylindrical Algebraic Decomposition*.

6.4 Saturation and Simplification

We now turn to the development of some proof procedures for both formula saturation and simplification. These procedures will generally accept as input a conjunctive quantifier-free **RCF** formula and will generate as output an equisatisfiable **RCF** formula in which "more useful" information has been made explicit. The idea is that with such information readily visible, techniques such as ICP and extended Tiwari methods may have an easier time recognising the satisfiability status of the original formula.

Of course, there exists a strong tension between increasing the amount of explicit information in a formula (so as to ease both machine and human understanding, aid further deductions, and so on), and the resulting formula size and complexity. Indeed, with an overly enthusiastic trigger finger, one can often saturate a formula with so much information that it becomes a sea of uninteresting data with key facts buried beneath the waves. Similarly, with over eager simplification, one might remove formally redundant ("subsumed") facts, even though their presence could make the operation of other proof procedures more effective. Resolving these tensions is in a sense the very heart of automated theorem proving.

Intuitively, one associates *saturation* with techniques which add derived facts to a formula, in contrast to *simplification* which should under some metric make a formula's salient features more succinctly represented. But, this distinction is often hard to justify, even though there is an obvious tension between the two. On the one hand, we have saturation adding information, while on the other, simplification attempts to reduce the noise. However, both have a common goal: To make the essential features of the facts clear. When a balance is struck through their careful combination, saturation and simplification can lead to inference mechanisms of remarkable power.

§

There is an important departure in this section from much of the prequel: When we reach the saturation rules built around multivariate polynomial factorisation, it will be essential to consider computer representational aspects of formulas, especially of terms. This is because multivariate factorisation will be a function between representations of the same abstract polynomial. That is to say, the explicit computer representation of a term, and hence of a formula, will matter. We have tried hard to be careful about the distinction between computer representation of terms and their abstract counterparts when it is essential to the task at hand, and have made pains to sweep it under the rug when it is not.

6.4.1 Saturation

Let us now present some saturation procedures which we have found useful. We will give each method in terms of inference rules which can be applied to atoms of a formula to generate derived facts.

6.4.1.1 Discriminants

The discriminant of a univariate polynomial p is a derived polynomial $\Delta(p)$ expressing fundamental properties of p's roots [GKW03]. For instance, $p \in \mathbb{Q}[x]$ has repeated complex roots if and only if $\Delta(p) = 0$. Moreover, if p is quadratic, then whether or not p has real roots can be decided from properties of $\Delta(p)$ alone. We will see how discriminants can be used to derive nontrivial facts not just from multivariate quadratic equations, but also from multivariate quadratic inequalities.

There is a natural description of the discriminant $\Delta(p)$ in terms of roots. For concreteness, we state it in the context of $\mathbb{Q}[x]$.

Definition 6.4.1 (Discriminant in terms of roots)**.** Let $p \in \mathbb{Q}[x]$ s.t. $p = \sum_{i=0}^{k} c_i x^i$ with $c_i \in \mathbb{Q}$. Then, the discriminant of p is

$$\Delta(p) = c_k^{2k-2} \prod_{i<j} (\eta_i - \eta_j)^2,$$

where $\eta_1, \ldots, \eta_k$ are the roots of p in some splitting field (counted with multiplicity).

This definition is nice as it makes the fundamental properties of the discriminant clear, but it is difficult to make use of algorithmically as one first needs to have a factorisation of p to compute it. Moreover, if one wishes to apply this definition to a multivariate polynomial which is seen as univariate with parameters (e.g., $p \in \mathbb{A}[x]$ with $\mathbb{A}$ a polynomial ring), then one is faced with the following difficulty: Galois theory shows us that beyond the quartic, one in general has no way to describe the individual roots of p exactly in terms of radicals. Thus, roots of p (to substitute for the η_i) need not be expressible as terms formed over $\mathbb{A}$ in the language of ordered rings, and so it is difficult to see how $\Delta(p)$ should even be a term which could appear in an **RCF** formula.

There is good news, however: A syntactic definition of the discriminant exists which, though it obfuscates the participation of the roots of p, expresses $\Delta(p)$ purely in terms of p's coefficients. It is straight-forward to compute and makes clear that even in the multivariate/parametric case, $\Delta(p)$ is always expressible in the language of ordered rings. This definition involves the concept of a polynomial *resultant*.

Definition 6.4.2 (Discriminant in terms of resultant)**.** Let $p \in \mathbb{A}[x]$ s.t. $p = \sum_{i=0}^{k} c_i x^i$ with $c_i \in \mathbb{A}[x]$. Then,

$$\Delta(p) = (-1)^{\frac{k(k-1)}{2}} \left(\frac{R(p, \frac{\partial p}{\partial x})}{c_k} \right),$$

where $R(p,q)$ is the resultant of p and q (w.r.t. x).

For convenience, we recall the definition of polynomial resultant, though we say nothing here about its remarkable properties.

Definition 6.4.3 (Resultant)**.** Let $p, q \in \mathbb{A}[x]$ s.t. $p = \sum_{i=0}^{k_1} c_i x^i$ and $q = \sum_{i=0}^{k_2} d_i x^i$ with

$c_i, d_i \in \mathbb{A}$. Then, the resultant of p and q (w.r.t. x)

$$R(p,q) = \begin{vmatrix} c_{k_1} & c_{k_1-1} & c_{k_1-2} & \dots & c_1 & c_0 & 0 & \dots & 0 \\ 0 & c_{k_1} & c_{k_1-1} & & \dots & & c_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \ddots & \ddots & 0 \\ 0 & \dots & 0 & c_{k_1} & c_{k_1-1} & \dots & & c_1 & c_0 \\ d_{k_2} & d_{k_2-1} & d_{k_2-2} & \dots & d_1 & d_0 & 0 & \dots & 0 \\ 0 & d_{k_2} & d_{k_2-1} & & \dots & & d_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \ddots & \ddots & 0 \\ 0 & \dots & 0 & d_{k_2} & d_{k_2-1} & \dots & & d_1 & d_0 \end{vmatrix}$$

is the determinant of the $(k_1+k_2)\times(k_1+k_2)$ *Sylvester matrix* of p and q (w.r.t. x)

Thus, given a multivariate polynomial $p \in \mathbb{Q}[x_1,\dots,x_n]$, we can compute the discriminant of p w.r.t. any x_i, resulting in a polynomial in $\mathbb{Q}[x_1,\dots,x_{i-1},x_{i+1},\dots,x_n]$. Let us write $\Delta(p,x_i)$ for this. Now we turn to how discriminants can be used as the basis of a saturation technique.

Consider φ a conjunctive $\exists$ **RCF** formula containing an atom of the form $(p=0)$ with $p \in \mathbb{Q}[\vec{x}]$. Let us assume p is quadratic in x_1, x_2 and x_3. Then, with a quick appeal to the quadratic formula, we see the following must be true:

$$\Delta(p,x_1) \geq 0 \ \wedge\ \Delta(p,x_2) \geq 0 \ \wedge\ \Delta(p,x_3) \geq 0.$$

This gives rise to the first discriminant saturation rule:

$$\text{QDS-EQ}\ \frac{ax_i^2+bx_i+c=0 \qquad a,b,c \in \mathbb{Q}[x_1,\dots,x_{i-1},x_{i+1},\dots,x_n]}{b^2-4ac \geq 0}$$

Though this may result in an inequality atom of the form $(b^2-4ac \geq 0)$ with b^2-4ac of higher multivariate total degree than the original equational assumption, the effects of this degree increase can be vastly outweighed by the gains one makes by having eliminated a variable.

Now, consider φ containing the conjunct $(p \leq 0)$. Then, less trivially, it will turn out that the following must be true:

$$(c_1 > 0 \implies \Delta(p,x_1) \geq 0) \wedge (c_2 > 0 \implies \Delta(p,x_2) \geq 0) \wedge (c_3 > 0 \implies \Delta(p,x_3) \geq 0),$$

where $c_i \in \mathbb{Q}[x_1,\dots,x_{i-1},x_{i+1},\dots,x_n]$ is the constant coefficient of p when p is seen as a univariate polynomial in $\mathbb{Q}[x_1,\dots,x_{i-1},x_{i+1},\dots,x_n][x_i]$.

This gives rise to the following saturation rule for conjunctive $\exists$ **RCF** formulas:

$$\textsf{QDS-LEQ}\ \frac{ax_i^2+bx_i+c\leq 0 \qquad c>0 \qquad a,b,c\in\mathbb{Q}[x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n]}{b^2-4ac\geq 0}$$

Lemma 6.4.4 (Correctness of QDS-LEQ)**.** *The correctness of rule* QDS-LEQ *is proved by verifying the following formula:*

$$\forall a,b,c\in\mathbb{R}\ \exists x\in\mathbb{R}\left(ax^2+bx+c\leq 0\ \wedge\ c>0 \implies b^2-4ac\geq 0\right).$$

Proof. Assume $\exists x(ax^2+bx+c\leq 0)$ and observe this is equivalent to $\neg\forall x(ax^2+bx+c>0)$. That is, $p(x)=ax^2+bx+c$ cannot be positive definite. But, by the Intermediate Value Theorem, $p(x)$ is positive definite iff the constant $p(0)=c$ is positive and $p(x)$ has no real roots. Expressing this in terms of c and the discriminant of $p(x)$, we have the equivalence:

$$\exists x(ax^2+bx+c\leq 0) \iff \neg(c>0\ \wedge\ b^2-4ac<0).$$

Thus, if $\exists x(ax^2+bx+c\leq 0)$ and $c>0$, then we must have that $b^2-4ac\geq 0$. □

This simple saturation rule has been for us at times very useful. For instance, it can often be used to recognise an unsatisfiable atom in a formula whose recognition one would expect to require multivariate factorisation or sums of squares decompositions. Let us make a small example.

Example 6.4.5. *Consider the atom*

$$4x^2-8xy+4y^2+1\leq 0$$

which we will examine as univariate in x, e.g.,

$$(4)x^2-(8y)x+(4y^2+1)\leq 0.$$

So, in the application of the rule, we have that

$$\begin{aligned} a &= 4\\ b &= 8y\\ c &= 4y^2+1 \end{aligned}$$

Since c is a trivial sum of squares with a positive constant, it will be automatically recognised to be strictly positive. Thus, we can apply the rule as follows (as $b^2-4ac=(8y)^2-4(4(4y^2+1))=64y^2-64y^2-16=-16$*):*

$$\textsf{QDS-LEQ}\ \frac{(4)x^2-(8y)x+(4y^2+1)\leq 0 \qquad 4y^2+1>0 \qquad 4,8y,4y^2+1\in\mathbb{Q}[y]}{-16\geq 0}$$

yielding immediately the proof of unsatisfiability by ground evaluation. And so we proved the multivariate atom $4x^2 - 8xy + 4y^2 + 1 \leq 0$ *to be unsatisfiable without ever having to realise that* $4x^2 - 8xy + 4y^2 + 1 = (2x - 2y)^2 + 1$.

And in the cases when these rules do not yield immediate proofs of unsatisfiability, they often derive restrictions on the variables which are useful for ICP, Tiwari and other methods.

There is an obvious dual to QDS-LEQ which we state now.

$$\text{QDS-GEQ}\ \frac{ax_i^2 + bx_i + c \geq 0 \qquad c < 0 \qquad a,b,c \in \mathbb{Q}[x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n]}{b^2 - 4ac \geq 0}$$

These ideas can in principle be extended to the quartic case as well. For instance, one could use the Descartes-Euler solution to compute the cubic resolvent and then exploit the fact that a quartic has no real roots if and only if all roots of the cubic resolvent are real with one positive and two negative, which can be determined by analysing two discriminants. But, the resulting **RCF** statements expressing real root existence in this way would quickly become astronomically large. We therefore do not make use of discriminant saturation explicitly beyond the quadratic, but we do employ degree reduction methods (covered shortly) to derive equivalent lower-degree **RCF** formulas from higher-degree ones when possible. Thus, quadratic discriminant saturation can be unexpectadly useful for formulas which on the surface seem to be beyond its reach.

6.4.1.2 Real Radical Ideal Approximations

Over $\mathbb{C}$, the correspondence between ideals and varieties is elucidated by Hilbert's Strong Nullstellensatz.

Theorem 6.4.6 (2)**.** *[Hilbert's Strong Nullstellensatz]*

$$\begin{aligned}\mathcal{I}(\mathcal{V}_{\mathbb{C}}(\mathcal{I}(\{p_1,\ldots,p_k\}))) &= \sqrt{\mathcal{I}(\{p_1,\ldots,p_k\})} \\ &= \left\{p \in \mathbb{Q}[\vec{x}] \mid \exists i \in \mathbb{N}\ s.t.\ p^i \in \mathcal{I}(\{p_1,\ldots,p_k\})\right\}.\end{aligned}$$

That is, given $p_i, q \in \mathbb{Q}[\vec{x}]$ the decision problem for universal Horn formulas over $\mathbb{C}$ can be reduced to an ideal membership check for radical ideals as follows:

$$\begin{aligned}\langle \mathbb{C}, +, -, *, 0, 1\rangle \models \left(\bigwedge_{i=1}^{k} p_i = 0\right) &\Longrightarrow q = 0 \\ &\Longleftrightarrow \\ q \in \sqrt{\mathcal{I}(\{p_1,\ldots,p_k\})}&\end{aligned}$$

which can then be effectively solved using Gröbner bases. Modulo ideal membership checking, the important step here is the construction, from a set of generators for an ideal $\mathfrak{I}$ over $\mathbb{Q}[\vec{x}]$, to a set of generators for the *radical ideal* containing $\mathfrak{I}$. This is a classically studied problem in algebraic geometry and most modern computer algebra systems provide efficient algorithms for complex ideal radicalization [Lap06].

Over $\mathbb{R}$, however, things are not so simple. The algebraic structure analogous to a radical ideal for real algebraic varieties, the so-called *real radical ideal*, has to take into account the order structure of $\mathbb{R}$ by incorporating polynomial summands that are sums of squares. That is, letting $\mathfrak{I} = \mathcal{I}(\{p_1, \ldots, p_k\})$,

$$\mathcal{I}(\mathcal{V}_{\mathbb{R}}(\mathfrak{I})) = \qquad \sqrt[\mathbb{R}]{\mathfrak{I}} = \qquad \left\{ p \in \mathbb{R}[\vec{x}] \mid p^{2i} + s \in \mathfrak{I} \mid s \in \sum (\mathbb{R}[\vec{x}])^2,\ i \in \mathbb{N} \right\}.$$

This has the analogous property over $\mathbb{R}$ that the classical radical ideal does over $\mathbb{C}$. Note of course that $\mathfrak{I} \subseteq \sqrt[\mathbb{R}]{\mathfrak{I}}$ as for any $p \in \mathfrak{I}$, we have $p^2 \in \mathfrak{I}$.

Known methods for transforming an ideal into its real radical are computationally infeasible for non-trivial problems, so we seek a method that *approximates* real radicalisation to obtain some practically useful membership decisions in an efficient way. This gives rise to the following saturation machinery whose correctness over $\mathbb{R}$ is immediate.

$$\textsf{RRI-GE}\ \frac{\bigwedge_{i=1}^{k} p_i = 0 \qquad (x^{2m} - q) \in \mathcal{I}(\{p_1, \ldots, p_k\}) \qquad \sqrt[2m]{q} \in \mathbb{Q}}{x = \sqrt[2m]{q} \ \vee\ x = -\sqrt[2m]{q}}$$

$$\textsf{RRI-GO}\ \frac{\bigwedge_{i=1}^{k} p_i = 0 \qquad (x^{2m+1} - q) \in \mathcal{I}(\{p_1, \ldots, p_k\}) \qquad \sqrt[2m+1]{q} \in \mathbb{Q}}{x = \sqrt[2m+1]{q}}$$

$$\textsf{RRI-VE}\ \frac{\bigwedge_{i=1}^{k} p_i = 0 \qquad (x^{2m_1} - y^{2m_2}) \in \mathcal{I}(\{p_1, \ldots, p_k\}) \qquad m_2 | m_1}{x^{m_1/m_2} = y \ \vee\ -x^{m_1/m_2} = y}$$

$$\textsf{RRI-VO}\ \frac{\bigwedge_{i=1}^{k} p_i = 0 \qquad (x^{2m_1+1} - y^{2m_2+1}) \in \mathcal{I}(\{p_1, \ldots, p_k\}) \qquad 2m_2+1 | 2m_1+1}{x^{2m_1+1/2m_2+1} = y}$$

Note (i) we usually restrict their use in practice to x and y being indeterminates, and (ii) the target terms (e.g., q in RRI-GE) need not be guessed, as if the antecedent holds, one can obtain q by reducing x^{2m} modulo $GB_{\prec}(\{p_1, \ldots, p_k\})$. This reduction process can be done incrementally for heuristically selected terms in a formula, with m ranging from 1 to some degree bound computed as a function of the generators of $GB_{\prec}(\{p_1, \ldots, p_k\})$. Observe that these rules really are saturating with equations corresponding to members of the *real* radical ideal containing $\{p_1, \ldots, p_k\}$, as they

would not hold over the complexes due to the existence of non-trivial roots of unity. For instance, over $\mathbb{C}$ it is false that

$$x^4 - 16 = 0 \implies (x = 2 \vee x = -2)$$

due to the existence of the non-trivial quartic roots of unity $e^{\pm\frac{\pi i}{2}}$. But, using rule RRI-GE we can derive over $\mathbb{R}$ that this fact does indeed hold, which corresponds to the fact that

$$x^2 - 4 \in \sqrt[\mathbb{R}]{\mathcal{I}(\{x^4 - 16\})},$$

since $x^2 - 4 = (x-2)(x+2)$.

6.4.1.3 Parametric Root Bounds

Let $p(x) \in \mathbb{A}[x]$ s.t. $p(x) = \sum_{i=0}^{k} c_i x^i$ with $c_i \in \mathbb{A}$. Let $\eta \in \mathbb{R}$ s.t. $p(\eta) = 0$. Then, a classical result of Cauchy tells us a bound on the absolute value of η in terms of the coefficients c_i. Precisely,

$$|\eta| \leq 1 + \frac{1}{|c_k|} \sum_{i=0}^{k-1} |c_i|.$$

This bound is straight-forward to calculate, but is difficult to apply as a saturation rule when $\mathbb{A} \neq \mathbb{Q}$. This is because absolute value is not primitive operation in the language of ordered rings and thus its effect on non-constant polynomials must be cumbersomely encoded (resulting in derived formulas which are usually not naturally expressed as conjunctions). It is, however, useful for atomic equations over $\mathbb{Q}[x]$ so that the computed bound is a rational number.

$$\text{RRB-GEQ}\ \frac{p(x) = 0 \qquad p(x) \in \mathbb{Q}[x] \qquad x \geq 0}{x \leq 1 + \frac{1}{|c_k|} \sum_{i=0}^{k-1} |c_i|}$$

$$\text{RRB-LEQ}\ \frac{p(x) = 0 \qquad p(x) \in \mathbb{Q}[x] \qquad x < 0}{-x \leq 1 + \frac{1}{|c_k|} \sum_{i=0}^{k-1} |c_i|}$$

$$\text{RRB-Q}\ \frac{p(x) = 0 \qquad p(x) \in \mathbb{Q}[x]}{x \geq (-1)\left(1 + \frac{1}{|c_k|} \sum_{i=0}^{k-1} |c_i|\right) \ \wedge\ x \leq \left(1 + \frac{1}{|c_k|} \sum_{i=0}^{k-1} |c_i|\right)}$$

Since the time of Cauchy, many other root bounds have been established. For our uses, the method of Kennedy in 1939 is especially nice as it yields parametric upper and lower bounds for real roots which are polynomials in the coefficient ring,

avoiding absolute values altogether [Ken39]. This leads to very simple saturation rules which can be applied parametrically when the polynomials of interest are multivariate. Let us present the Kennedy bounds for cubic, quartic and degree six polynomials followed by saturation rules based upon them. We do not know of any extension of the method Kennedy used to obtain these bounds to polynomials of degree higher than six. For higher degree polynomials which are actually univariate, i.e., in $\mathbb{Q}[x]$, the Cauchy bounds mentioned previously may of course be applied.

First, the cubic. Let $p(x) = x^3 + a_2x^2 + a_1x + a_0$ s.t. $p(x) = 0$ and $a_2 \neq 0$. Then, Kennedy shows the following:

$$\begin{aligned} a_2 < 0 &\implies x^3 \geq \frac{a_1^2}{4a_2} - a_0 \\ a_2 > 0 &\implies x^3 \leq \frac{a_1^2}{4a_2} - a_0 \end{aligned}$$

Next, the quartic. Let $p(x) = x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ s.t. $p(x) = 0$ and $a_3 \neq 0$. Then,

$$\begin{aligned} a_3 < 0 &\implies x^3 \geq \frac{1}{4a_3}(a_2^2 + a_1^2 - 4a_0) \\ a_3 > 0 &\implies x^3 \leq \frac{1}{4a_3}(a_2^2 + a_1^2 - 4a_0) \end{aligned}$$

Finally, the sixth degree. Let $p(x) = x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ s.t. $p(x) = 0$ and $a_5 \neq 0$. Then,

$$\begin{aligned} (a_5 < 0 \wedge a_4 > 0) &\implies x^3 \geq \frac{1}{4a_5}\left(a_3^2 + \frac{a_2^2}{a_4} + a_1^2 - 4a_0\right) \\ (a_5 > 0 \wedge a_4 > 0) &\implies x^3 \leq \frac{1}{4a_5}\left(a_3^2 + \frac{a_2^2}{a_4} + a_1^2 - 4a_0\right) \end{aligned}$$

Let us adapt these bounds to appropriate saturation rules. Note how each degree's pair of implications collapses into a single rule.

$$\text{RRB-K3}\ \frac{x^3 + a_2x^2 + a_1x + a_0 = 0 \qquad a_2 \neq 0}{4a_2x^3 \leq a_1^2 - 4a_2a_0}$$

$$\text{RRB-K4}\ \frac{x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0 \qquad a_3 \neq 0}{4a_3x^3 \leq (a_2^2 + a_1^2 - 4a_0)}$$

$$\text{RRB-K6}\ \frac{x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0 \qquad a_4 > 0 \qquad a_5 \neq 0}{4a_4a_5x^3 \leq a_2^2 + a_4\left(a_3^2 + a_1^2 - 4a_0\right)}$$

6.4.1.4 Orientations

This class of saturation rules has a simple purpose: To provide multiple orientations of the same atom. For instance, when an atom contains a polynomial with multiple linear monomials (e.g., monomials consisting of a product of a rational number and a single indeterminate), it will often be convenient to have variations of the atom with each having a different single indeterminate on the left-hand side. This is especially useful for our ICP calculus (cf. **Figure 6.1**), where contraction strength is sensitive to the orientations of the constraint hypotheses. In the calculus, providing multiple orientations of the same atom can at worst do nothing to the interval context and at best decrease the size of intervals known to contain terms, increasing knowledge of the solution space and moving one closer to obtaining an empty interval (unsatisfiability) judgment.

To ease the expression of these rules, we will use the sum-of-monomials representation of polynomials introduced in **Section 2.3**. Recall that in this notation, $\mathcal{E} \subset \mathbb{N}^n$ and $c_\alpha \in \mathbb{Q}$. Let us use $L_k(\alpha)$ to mean that α is an exponent vector s.t. $\vec{x}^\alpha = x_k$, i.e.,

$$L_k(\alpha) \iff \alpha(k) = 1 \wedge \forall 1 \leq i \leq n\,(i \neq k \implies \alpha(k) = 0)\,.$$

$$\text{ORI-EQ}\ \frac{\sum_{\beta\in\mathcal{E}} c_\beta \vec{x}^\beta = 0 \qquad L_k(\alpha) \qquad c_\alpha \neq 0}{x_k = (-\frac{1}{c_\alpha})\sum_{\beta\in\mathcal{E}\setminus\{\alpha\}} c_\beta \vec{x}^\beta}$$

$$\text{ORI-GEQ-G}\ \frac{\sum_{\beta\in\mathcal{E}} c_\beta \vec{x}^\beta \geq 0 \qquad L_k(\alpha) \qquad c_\alpha > 0}{x_k \geq (-\frac{1}{c_\alpha})\sum_{\beta\in\mathcal{E}\setminus\{\alpha\}} c_\beta \vec{x}^\beta}$$

$$\text{ORI-GEQ-L}\ \frac{\sum_{\beta\in\mathcal{E}} c_\beta \vec{x}^\beta \geq 0 \qquad L_k(\alpha) \qquad c_\alpha < 0}{x_k \leq (-\frac{1}{c_\alpha})\sum_{\beta\in\mathcal{E}\setminus\{\alpha\}} c_\beta \vec{x}^\beta}$$

$$\text{ORI-G-G}\ \frac{\sum_{\beta\in\mathcal{E}} c_\beta \vec{x}^\beta > 0 \qquad L_k(\alpha) \qquad c_\alpha > 0}{x_k > (-\frac{1}{c_\alpha})\sum_{\beta\in\mathcal{E}\setminus\{\alpha\}} c_\beta \vec{x}^\beta}$$

$$\text{ORI-G-L}\ \frac{\sum_{\beta\in\mathcal{E}} c_\beta \vec{x}^\beta > 0 \qquad L_k(\alpha) \qquad c_\alpha < 0}{x_k < (-\frac{1}{c_\alpha})\sum_{\beta\in\mathcal{E}\setminus\{\alpha\}} c_\beta \vec{x}^\beta}$$

There are obvious dual $\leq,<$ inequality rules ORI-LEQ-G, ORI-LEQ-L, ORI-L-L, ORI-L-G one can define (though of course the above rules are sufficient for $\leq,<$ by simply multiplying the original atom through by -1).

6.4.1.5 Factorisations

As discussed previously, when faced with a multivariate atom, it is often useful to have multiple representations of the polynomials appearing in it. Factorisations of polynomials can be especially illuminating when it comes to understanding the effect an atom will have on the solution space of a formula in which it appears.

Chiefly, multiple representations of the same polynomial can enhance the effectivity of ICP by allowing one to further tighten intervals. This can help one attack the so-called *dependency problem* of ICP, which often leads to vast over-estimations of containing intervals [Krä06]. Outside of ICP, factorisations have much inferential utility, e.g., fully-factored representations often allow one to easily infer nontrivial sign constraints on a polynomial using only light-weight reasoning similar to that we used for recognising state witnesses in the original Tiwari method. Moreover, knowing factorisations and other representations of the same polynomial can enhance the power of simplification methods we will encounter shortly.

To exploit factorisations in this way, we must deal with computer representation of our terms. This is so that we can distinguish between two different representations of

the same abstract polynomial. Then, for instance, when given the term

```
(+ (* X (* X (* X X)))
   (+ (* -2 (* K (* X (* X X))))
      (+ (* (* K K) (* X X))
         (+ (* 6 (* W (* X (* X X))))
            (+ (* -6 (* W (* K (* X X))))
               (+ (* 9 (* (* W W) (* X X)))
                  (+ (* -4 (* Z (* X (* X X))))
                     (+ (* 4 (* Z (* K (* X X))))
                        (+ (* -12 (* Z (* W (* X X))))
                           (* 4 (* (* Z Z) (* X X)))))))))))))
```

it can be factored it into the nicer representation

```
(* (EXPT (- K (+ (- X (* 2 Z)) (* 3 W))) 2) (EXPT X 2))
```

which is then easily recognised to be non-negative simply by analysing the power arguments of the exponentation operations. We will discuss more of these practical representation concerns in the sequel. For now, it is enough to realise that the factorisation procedure is a function between computer representations of the same polynomial.

Efficient algorithms for multivariate polynomial factorisation and the closely related problem of multivariate GCD are difficult and profound pillars of computer algebra. Modern approaches, such as those based on Hensel Lifting [Kal85] and ideas related to EZGCD [MY73], benefit much from hybrid implementations in which the core algorithms have been substantially enhanced by a multititude of ad hoc heuristics and special tricks. Factorisation and GCD algorithms found in mainstream computer algebra packages, for instance, consist generally of complex ad hoc combinations of core modern algorithms and such heuristics, with the exact brew of techniques used in closed commercial systems usually (most unfortunately) a closely guarded secret.

Thus, unlike most techniques discussed in this thesis, we have decided not to implement multivariate factorisation ourselves. Instead, we have integrated our tool **RAHD** with the open-source computer algebra system **Maxima** [LR08], and use its functionality for multivariate factorisation (and GCD, square-free decompositions and more). As both tools are written in Common Lisp, their integration is seamless. **Maxima**, a direct descendent of the immeasurably impactful **MACSYMA** system, is powerful and robust, has a large active community of users, and most importantly, consists of open, well-documented source code.

Of course, no trust need be lost when delegating factorisation to an external procedure, as one simply verifies by multiplication that the factorisation is correct before believing it.

In the rule below, we will use `P` and `Q` to mean computer representations of the polynomials p and q (in $\mathbb{Q}[\vec{x}]$). Two computer terms will correspond to the same abstract polynomial if and only if their expansions into a computer representation of sum-of-monomials normal form are identical. Let us clarify this fact by a brief representation digression.

In **RAHD**, we primarily use two different internal representations of polynomials:

1. A "human readable" representation in which terms are given as Lisp S-expressions over Lisp rational numbers, variable symbols, and the binary arithmetic operations (`+, -, *, EXPT`).

2. An "algebraic" representation corresponding to sum-of-monomials normal form. In this representation, polynomials are given as a set of monomials, each of which is a pair of a rational number and a power product, with a power-product being a set of variable power pairs, and a variable power pair being a pair of a variable symbol and a natural number greater than 1. When a monomial order is used to arrange the members, each of these sets can be implemented as a list with the nice property that two lists shall be identical if and only if they correspond to the same set, and thus, the same polynomial. Hence, given a monomial order, each polynomial will have a unique computer representation in this sum of monomials form.

We call the first representation "extended `EXPT`," as an exponentiation operator is not officially a part of the language of ordered rings. Of course, we only use `EXPT` when its power argument is a fixed natural number, so everything expressible in this notation is expressible in the language of ordered rings directly. It's just that having `EXPT` around makes reasoning about the signs of factorisations easier.

We call the second representation "algebraic," as this is the representation used, for instance, by the Gröbner basis construction algorithms. In what follows, we assume we have fixed some monomial order so that this each abstract polynomial has a unique algebraic computer representation.

Let $\mathbb{T}_{\mathbb{E}}$ be the set of computer terms in extended `EXPT` representation and let $\mathbb{T}_A$ be the set of computer terms in algebraic representation. Then,

$$Expand : \mathbb{T}_E \to \mathbb{T}_A$$

will be the obvious surjection translating an extended EXPT term into its computer sum-of-monomials normal form, and

$$Poly : \mathbb{T}_A \to \mathbb{Q}[\vec{x}]$$

will be the obvious bijection associating a computer sum-of-monomials normal form with its abstract counterpart. Then,

$$Factor : \mathbb{T}_E \to \mathbb{T}_E$$

will be s.t.

$$\forall \mathtt{P} \in \mathbb{T}_E \quad Expand(Factor(\mathtt{P})) = Expand(\mathtt{P}).$$

We will not place any more constraints upon the factorisation algorithm than this. This is for pragmatic reasons: Multivariate factorisation is such a difficult problem that, as we stated earlier, most high-powered implementations combine complete algorithms with a multitude of heuristics and tricks for special cases. Usually, these implementations (including that of **Maxima**) allow the user to give an argument expressing roughly how hard the system should try to obtain a full factorisation. When the system deems this bound has been exceeded, it may return a term which, though correct in the sense of corresponding to the same abstract polynomial as the original, is only partially factored. Even these partial factorisations can often be very useful and we want to allow them.

Finally, to minimise cognitive dissonance, we will present the atoms in the saturation rules based upon factorisation using their computer representation. A computer atom will be an S-expression of the form

```
(_R_ P Q)
```

where the relation symbol _R_ is drawn from

$$\{<, <=, =, >=, >\}.$$

Then, the simple factorisation rule is as follows.

$$\textsf{FactorAtom}\ \frac{\texttt{(\_R\_ P Q)} \qquad \texttt{\_R\_} \in \{<, <=, =, >=, >\}}{\texttt{(\_R\_}\ Factor(\mathtt{P})\ Factor(\mathtt{Q})\texttt{)}}$$

It is convenient to have a rule which combines the zeroing of an atom's right-hand side, factors the resulting left-hand side, and then attempts to deduce sign information

about the left-hand polynomial. The FACTOR-SIGN rule is designed for this. It will use a function called *DeduceSign* of the form

$$DeduceSign : \mathbb{T}_E \to \{<, <=, =, >=, >\} + \{\texttt{NIL}\}$$

s.t.

$$\forall \texttt{P} \in \mathbb{T}_E \quad [(DeduceSign(\texttt{P}) = \texttt{\_R\_} \neq \texttt{NIL}) \implies (\forall \vec{r} \in \mathbb{R}^n \ [Poly(Expand(\texttt{P}))(\vec{r}) \odot_{\texttt{\_R\_}} 0])],$$

where $\odot_{\texttt{\_R\_}}$ is the mathematical equality or inequality relation corresponding to `_R_`. For example,

$$DeduceSign(\texttt{(* (EXPT (- K (+ (- X (* 2 Z)) (* 3 W))) 2) (EXPT X 2))})$$

will be >=. We include the Common Lisp source code of *DeduceSign* in the resource given in **Appendix A** as the reasoning involved is as tedious as it is pedestrian.

Finally, the FACTOR-SIGN rule.

$$\text{FACTOR-SIGN} \ \frac{\texttt{(\_R\_ P Q)} \qquad S = Factor(\texttt{(- P Q)}) \qquad \texttt{\_D\_} = DeduceSign(S) \neq \texttt{NIL}}{\texttt{(\_D\_ (- P Q) 0)}}$$

There are many other uses for multivariate factorisations. We will return to them shortly when we examine term and formula simplification.

6.4.2 Simplification

We now turn to mechanisms for formula simplification. Formula simplification is an area of both monumental depth and breadth; a true cornerstone of mathematics as a whole, and a central focus of computer algebra and automated reasoning. Indeed, the entire enterprise of mechanical theorem proving can be seen as an exercise in formula simplification. But then, so can the evaluation of any partial recursive function.

In this section, we do not even attempt to give a global contextual view of formula simplification and its tremendous generality. Instead, we simply present a small catalogue of simplification techniques which we have found useful during ∃ **RCF** decision making. Our goal is to make clear the mathematics behind the methods we actually use. We will discuss their application in the sequel.

§

When it comes to formula simplification in the context of **RCF** quantifier elimination, the 2000 PhD thesis by Andreas Dolzmann [Dol00], *Algorithmic Strategies for Applicable Real Quantifier Elimination*, is a remarkable achievement.

His thesis is focused upon formula simplification during the execution of a quantifier elimination method known as *virtual term substitution* [Wei97]. However, it is a *tour de force* of simplification, and much of what he says is applicable in our more general setting of combined **RCF** decision methods. We make use of a number of the simplification mechanisms he presents, e.g., those based on square-free parts, parity decompositions and degree shifts.

Before giving an account of these and other simplification methods, let us be so bold as to quote part of his introduction (**p16, Section 2.2**). His main point is that there are many conflicting metrics under which one can measure formula complexity, and thus, many incompatible metrics under which one can measure simplification progress.

> **[Begin Quotation — Dolzmann PhD : p16, Section 2.2]**
> **The Notion of Simple Formulas**
>
> It is not obvious which formulas should be considered simple. We summarise some *simplification goals*:
>
> - **Few atomic formulas** Currently, this is our main goal. Quantifier elimination output is in general too large to be understood by a human. However, it is often small enough for applications where it is processed automatically, typically by repeatedly fixing the values of some variables and then eveluating by resimplification. Small formulas then minimize memory consumption and evaluation time.
> - **Comprehensible boolean structure** When using quantifier elimination as a tool for solving mathematical problems it is essential that the output is comprehensible. Examples for comprehensible boolean structures are comparatively flat formulas or case distinctions.
> - **Few different atomic formulas** This is convenient for quantifier elimination by [virtual term substitution]. In addition, it supports many simplification strategies.
> - **Simple terms** We consider it unintuitive when information that can be encoded logically is actually encoded algebraically. For instance, we would prefer the disjunction $a = 0 \vee b = 0$ to the product $ab = 0$.
> - **Small satisfaction sets** of the contained atomic formulas. This leads to a formula that is less redundant. If we know e.g. that $a \neq 0$ for some reason, we can [perhaps] replace $a \neq 0$ by [for instance] $a < 0$, which has a smaller satisfaction set.
> - **Convenient relations** For [virtual term substitution], weak orders are more convenient than strong ones. On the other hand, equations and disequations can be considered simpler than orders.

- **Convenient boolean operations** We consider conjunction and disjunction to be simpler than implication, replication, and equivalence.

Some of the simplification goals given above contradict one another. [...]
[End Quotation]

In our general setting of combining a heterogeneous collection of $\exists$ **RCF** proof procedures, we have to deal with many of these same simplification conflicts. From a high level, our answer is to provide as many different simplification mechanisms as possible, with many of them pairwise disagreeing on what makes one formula simpler than another, and to then allow the user to strategically combine them as he sees fit. Many of these simplification methods are further parameterised by simplification seed data which can be user-specified, e.g., Gröbner basis simplification methods are parameterised by a monomial order, which itself is parameterised by a variable order.

In what follows, we often assume the RHS of atoms have been zeroed by subtraction of the RHS from both sides. Similarly, it is often convenient to assume atoms with polynomials in $\mathbb{Q}[\vec{x}]$ have been converted into equivalent atoms with polynomials in $\mathbb{Z}[\vec{x}]$. We will present simplification rules in a deductive form, in a similar format to the saturation rules given previously. Note though that these simplification rules are about replacing one subformula with another "simpler" one, not about adding to some pool of known facts as with saturation.

6.4.2.1 Evaluation, Arithmetic Simplification and Directed Rewriting

These simplification rules perform three simple tasks:

1. The evaluation of ground[6] atoms,

2. The arithmetic simplification of terms, both in a light-weight form and in a heavy-weight form which canonicalises terms into sum-of-monomials normal form respecting a monomial order,

3. The use of orientations and a monomial ordering to derive a set of terminating rewrite rules from polynomial equations which contain linear monomials, followed by their application.

[6]Note that in this dissertation, an atom is called *ground* iff its terms contain only arithmetical combinations of rational numbers. This is a bit contentious, as since our formulas are all implicitly existentially quantified, every "variable" can really be taken to be a Skolem constant, and thus *every* term in our formulas can be seen to be ground in the sense usually used in mathematical logic. Nevertheless, given our restriction to $\exists$ **RCF**, our usage is intuitive and convenient.

In practice, task 3 often leads to atoms which can be simplified via tasks 1 and 2. Rewrite rules derived in task 3 always lead to the elimination of a variable, which can be very useful before applying proof methods whose complexity is heavily dependent upon formula dimension, e.g., cylindrical algebraic decomposition. Both, tasks 1 and 2 can lead to variable elimination as well, e.g., if a variable only appears in a term in which it is multiplied by zero.

Ground evaluation is handled by the following rule, where $Ground(\texttt{P})$ holds iff `x` is a ground term, $EvalAtom(\texttt{(\_R\_ P 0)})$ evaluates a ground atom using exact rational arithmetic.

$$\text{SIMP-GEVAL}\ \frac{\texttt{(\_R\_ P 0)} \qquad Ground(\texttt{P})}{EvalAtom(\texttt{(\_R\_ P 0)})}$$

Now, when a term is presented as part of an atom, it can easily be that the term is not in sum-of-monomials normal form. This is especially true when problems of a geometrical or physical nature are presented by hand, and also occurs often after derived term manipulations have been applied to intermediate formulas, e.g., after the application of rewrite rules.

Many proof procedures or components thereof, such as cylindrical algebraic decomposition or Gröbner basis computation (including the Tiwari procedure), will first canonicalise polynomials into sum-of-monomials normal form before they begin their processing. This normalisation causes many types of arithmetical simplifications to take place. But, in doing so, the size of terms can grow tremendously and important information about the underlying polynomials can easily become hidden.

Therefore, it is prudent to have arithmetic simplification machinery which does *not* perform this normalisation. Instead, this machinery should apply minimal transformations which attempt to eliminate trivial arithmetical components of terms while still respecting as much as possible the original structure of the term presentation. We call this *light-weight arithmetical simplification*. To perform it, we need to work recursively over the tree structure of a computer term representation. Then, light-weight arithmetical simplification is achieved by the following simple rewrite rules (with `P` matching any term) together with a rule for evaluating ground subterms:

- `(- P P)` $\mapsto$ `0`,
- `(+ P (- 0 P))` $\mapsto$ `0`,

- `(+ (- 0 P) P)` $\mapsto$ `0`,
- `(* P 0)` $\mapsto$ `0`,
- `(* 0 P)` $\mapsto$ `0`,
- `(+ P 0)` $\mapsto$ `P`,
- `(+ 0 P)` $\mapsto$ `P`,
- `(- P 0)` $\mapsto$ `P`,
- `(* P 1)` $\mapsto$ `P`,
- `(* 1 P)` $\mapsto$ `P`,
- `(* (- 0 P) (- 0 P))` $\mapsto$ `(* P P)`,
- `(EXPT P 0)` $\mapsto$ `1`,
- `(EXPT P 1)` $\mapsto$ `P`.

While working recursively over a term, we also want to evaluate any ground subterms. This is done by the following conditional rule with $EvalTerm(\texttt{P})$ evaluating a ground term:

$$Ground(\texttt{P}) \implies \texttt{P} \mapsto EvalTerm(\texttt{P}).$$

Then, letting *LightSimp* be a function which traverses the term structure and applies the above rules until a fixed point is reached, we encapsulate this light-weight simplification into the following rule.

$$\text{SIMP-ARITH-LW}\ \frac{(\texttt{\_R\_}\ \texttt{P}\ \texttt{Q}) \qquad \texttt{\_R\_} \in \{\texttt{<}, \texttt{<=}, \texttt{=}, \texttt{>=}, \texttt{>}\}}{(\texttt{\_R\_}\ LightSimp(\texttt{P})\ LightSimp(\texttt{Q}))}$$

When light-weight simplification is insufficient, heavy-weight simplification may be used. This involves the function $Expand : \mathbb{T}_E \to \mathbb{T}_A$ introduced in **Section 6.4.1.5** which maps a term in extended EXPT notation into its computer sum-of-monomial normal form, which is itself governed by an active monomial ordering (left implicit). In the process, the RHS is zeroed.

$$\text{SIMP-ARITH-HW}\ \frac{\texttt{(\_R\_ P Q)} \qquad \texttt{\_R\_} \in \{\texttt{<}, \texttt{<=}, \texttt{=}, \texttt{>=}, \texttt{>}\}}{\texttt{(\_R\_}\ \mathit{Expand}(\texttt{(- P Q)})\ \texttt{0)}}$$

Finally, we turn to a simple mechanism for directed rewriting. This extracts a rewrite rule from an equation `(= P Q)` if the canonicalised polynomial $\mathit{Expand}(\texttt{(- P Q)})$ of the equation contains a monomial in a single indeterminate `X` s.t. `X` does not appear in any other monomial in $\mathit{Expand}(\texttt{(- P Q)})$. An active monomial order is used so that if multiple such monomials exist in the canonicalised polynomial of the equation (each in a different indeterminate), only the largest one is used as the source of a rewrite rule. This has the nice effect that one can apply this process to all equations in a formula, and the resulting system of rewrite rules will be guaranteed to be terminating. Moreover, one can iterate this process by deriving the terminating system of rewrite rules, applying them as to obtain a formula in one less indeterminate, and then attempting the whole process again upon the result.

We wrap this up as the following rule, where $\mathit{ExtractR}(\texttt{S})$ extracts a rewrite rule from `S` as above and $\mathit{ApplyR}(R, \texttt{U})$ applies the rewrite rule R to the term `U`. As before, we let $\texttt{\_R\_} \in \{\texttt{<}, \texttt{<=}, \texttt{=}, \texttt{>=}, \texttt{>}\}$.

$$\text{SIMP-LRW}\ \frac{\texttt{(= P Q)} \qquad \texttt{(\_R\_ W V)} \qquad R = \mathit{ExtractR}(\mathit{Expand}(\texttt{(- P Q)})) \quad R \neq \mathit{NIL}}{\texttt{(\_R\_}\ \mathit{ApplyR}(R,\ \texttt{W})\ \mathit{ApplyR}(R,\ \texttt{V})\texttt{)}}$$

6.4.2.2 Square-free Parts and Parity Decompositions

Let $p \in \mathbb{Z}[\vec{x}]$. Then, p is *square-free* if p has no divisor of multiplicity greater than 1. The *square-free decomposition*[7].

$$\langle p_1, \ldots, p_k \rangle \ \text{ s.t. } \ \prod_{i=1}^{k} p_i^i = p$$

with each p_i square-free and each pair of non-identical polynomials relatively prime (i.e., $p_i \neq p_j \implies (p_i, p_j) = 1$). The product $\prod_{i=1}^{k} p_i$ is called the *square-free part* of

[7]The definition of a square-free decomposition can be confusing when one first encounters it, as one might naively expect the product part of the definition to be of the form " $\langle p_1, \ldots, p_k \rangle$ s.t. $\prod_{i=1}^{k} p_i^{d_i}$ " for some indexed family of degrees d_i instead of " $\prod_{i=1}^{k} p_i^i$." But, once one realises this is not a typo, then it is not hard to see how a square-free factorisation can always be put into this canonical form. A nice reference for square-free decompositions and their algorithmic foundation is [Yun76].

p. Given a square-free decomposition of p as above, the *parity decomposition* of p is defined as a pair of the form

$$\langle \prod_{i=1,i \text{ odd}}^{k} p_i, \prod_{i=2,i \text{ even}}^{k} p_i \rangle.$$

A simplification mechanism due to Dolzmann utilises the following equivalences [Dol00].

Observation 6.4.7. Let $p \in \mathbb{Z}[\vec{x}]$, P be the square-free part of p and $\langle p_o, p_e \rangle$ the parity decomposition of p.

- $p = 0 \iff P = 0 \iff (p_o = 0 \vee p_e = 0)$,
- $p \neq 0 \iff P \neq 0 \iff (p_o \neq 0 \wedge p_e \neq 0)$,
- $p > 0 \iff p_o p_e^2 > 0 \iff (p_o > 0 \wedge p_e \neq 0)$,
- $p \geq 0 \iff p_o p_e^2 \geq 0 \iff (p_o \geq 0 \vee p_e = 0)$,
- $p < 0 \iff p_o p_e^2 < 0 \iff (p_o < 0 \wedge p_e \neq 0)$,
- $p \leq 0 \iff p_o p_e^2 \leq 0 \iff (p_o \leq 0 \vee p_e = 0)$.

These equivalences give rise to the following simplification rules.

$$\text{SIMP-PAR-EQ}\ \frac{p = 0 \qquad \mathit{ParityDecomp}(p) = \langle p_o, p_e \rangle}{(p_o = 0 \ \vee \ p_e = 0)}$$

$$\text{SIMP-PAR-NEQ}\ \frac{p \neq 0 \qquad \mathit{ParityDecomp}(p) = \langle p_o, p_e \rangle}{(p_o \neq 0 \ \wedge \ p_e \neq 0)}$$

$$\text{SIMP-PAR-G}\ \frac{p > 0 \qquad \mathit{ParityDecomp}(p) = \langle p_o, p_e \rangle}{(p_o > 0 \ \wedge \ p_e \neq 0)}$$

$$\text{SIMP-PAR-GEQ}\ \frac{p \geq 0 \qquad \mathit{ParityDecomp}(p) = \langle p_o, p_e \rangle}{(p_o \geq 0 \ \vee \ p_e = 0)}$$

$$\text{SIMP-PAR-L}\ \frac{p < 0 \qquad \mathit{ParityDecomp}(p) = \langle p_o, p_e \rangle}{(p_o < 0 \ \wedge \ p_e \neq 0)}$$

$$\text{SIMP-PAR-LEQ}\ \frac{p \leq 0 \qquad \mathit{ParityDecomp}(p) = \langle p_o, p_e \rangle}{(p_o \leq 0 \ \vee \ p_e = 0)}$$

6.4.2.3 PD, PSD and Trivial Sums of Squares

These simplification rules allow us to simplify an atom when we know its constituent polynomial is either positive definite (PD), positive semidefinite (PSD), negative definite (ND) or negative semidefinite (NSD). In some cases, we will be able to use the rules given above using square-free or parity decompositions to enhance this simplification.

Recall that a polynomial p is PSD iff it is strictly non-negative (i.e., only takes on non-negative values when given any real numbers for its variables), PD if it is strictly positive, NSD iff it is strictly non-positive, and ND iff it is strictly negative.

The $\exists$ **RCF** decision problem is simply reducible to the problem of deciding whether or not a multivariate polynomial in $\mathbb{Z}[\vec{x}]$ is PD [PdM09b]. That is to say, it is very difficult, and any exhaustive check of this nature is not what one would want to have as part of a fast simplification loop. But, there is one simple class of polynomials for which deciding negative/positive (semi-)definiteness is simple: the *trivial sums of*

squares. We have essentially seen this class of polynomials before during our definition of *state witnesses* in the original Tiwari method. That is, a polynomial p is a *trivial sum of squares* (TSOS) if when p is expressed in sparse sum-of-monomials normal form, every variable appearing in a monomial in p has even power and every coefficient is non-negative. A polynomial p is a *strict TSOS* if p is TSOS and p has a positive constant monomial.

With this in mind, let us recount simplification machinery again due to Dolzmann [Dol00]. It begins by the following simple observations.

Observation 6.4.8. Let $\langle p_o, p_e \rangle$ be the parity decomposition of $p \in \mathbb{Z}[\vec{x}]$. Then, p is PSD if p_o is TSOS. Furthermore, p is PD if both p_o and p_e are strict TSOS. (These implications do not follow in the other direction.)

Observation 6.4.9. Let $p \in \mathbb{Z}[\vec{x}]$ be PD. Then, we have the following equivalences:

1. $p = 0 \iff p < 0 \iff p \leq 0 \iff$ **false**,
2. $p \neq 0 \iff p > 0 \iff p \geq 0 \iff$ **true**.

Observation 6.4.10. Let $p \in \mathbb{Z}[\vec{x}]$ be PSD. Then, we have the following equivalences:

1. $p < 0 \iff$ **false**,
2. $p \geq 0 \iff$ **true**,
3. $p > 0 \iff p \neq 0$,
4. $p \leq 0 \iff p = 0$.

Note that by rules SIMP-PAR-EQ and SIMP-PAR-NEQ, the final two equivalences can be further extended as follows (with $\langle p_o, p_e \rangle$ the parity decomposition of p):

$$p > 0 \iff p \neq 0 \iff (p_o \neq 0 \wedge p_e \neq 0),$$

$$p \leq 0 \iff p = 0 \iff (p_o = 0 \vee p_e = 0).$$

By observing the following closure properties of (strict) TSOS polynomials, we will be able to further exploit the above equivalences.

Observation 6.4.11. Let $p_1, \ldots, p_k \in \mathbb{Z}[\vec{x}]$ each be TSOS. Then,

- $\prod p_i$ is TSOS,

- $\prod p_i$ is strict TSOS iff all p_i are strict TSOS,
- $\sum p_i$ is TSOS,
- $\sum p_i$ is strict TSOS if at least one p_i is strict TSOS.

We also have the more general observations on sums and products of PSD and PD polynomials:

Observation 6.4.12. Let $p_1, \ldots, p_k \in \mathbb{Z}[\vec{x}]$ each be PSD. Then,

- $\prod p_i$ is PSD,
- $\prod p_i$ is PD iff all p_i are PD,
- $\sum p_i$ is PSD,
- $\sum p_i$ is PD if at least one p_i is PD.

These observations are convenient, as they allow us to deduce a product or sum of polynomials to be PD, PSD (or ND, NSD) if we know the relevant facts about its factors. This is useful when polynomial terms are presented in a non-normalised form, e.g., with the product of polynomials written explicitly. Instead of immediately performing polynomial multiplication and normalising the product into a sum-of-monomials normal form (as would be done if we were to form a Gröbner basis or begin computing a cylindrical algebraic decomposition with the polynomials), we can first examine the factors and attempt to deduce sign information from them, carrying this information to the normalised product if we are successful.

Finally, the Dolzmann method makes additional use of the fact that TSOS polynomials can often be usefully split.

Observation 6.4.13. When p was PSD, **Observation 6.4.10** showed that $(p < 0)$ and $(p \geq 0)$ could both be decided, but $(p > 0)$ and $(p \leq 0)$ were only reduced to $(p \neq 0)$ and $(p = 0)$ (resp.). In these cases, we can use the following two equivalences to split the constraint involving a TSOS polynomial $p = \sum s_i$ as follows:

$$\left(\sum s_i \leq 0\right) \iff \left(\sum s_i = 0\right) \iff \left(\bigwedge s_i = 0\right),$$

$$\left(\sum s_i > 0\right) \iff \left(\sum s_i \neq 0\right) \iff \left(\bigvee s_i \neq 0\right).$$

These observations give rise to the following simplification rules. We use $PSD(p)$ (resp. $PD(p)$) to mean that we have proven p to be PSD (resp. PD). This may be because p is TSOS (resp. strict TSOS), but this fact could also have been deduced by other means. Of course, if $PD(p)$ is known, then $PSD(p)$ is known. We use $TSOS(p, \sum s_i)$ to mean that we have recognised p to be a trivial sum of squares with each s_i a trivial square s.t. $p = \sum s_i$.

$$\text{SIMP-PD-EQ}\ \frac{p = 0 \qquad PD(p)}{\textbf{false}}$$

$$\text{SIMP-PD-LEQ}\ \frac{p \leq 0 \qquad PD(p)}{\textbf{false}}$$

$$\text{SIMP-PSD-L}\ \frac{p < 0 \qquad PSD(p)}{\textbf{false}}$$

$$\text{SIMP-PD-NEQ}\ \frac{p \neq 0 \qquad PD(p)}{\textbf{true}}$$

$$\text{SIMP-PSD-GEQ}\ \frac{p \geq 0 \qquad PSD(p)}{\textbf{true}}$$

$$\text{SIMP-PD-G}\ \frac{p > 0 \qquad PD(p)}{\textbf{true}}$$

$$\text{SIMP-PSD-G}\ \frac{p > 0 \qquad PSD(p)}{p \neq 0}$$

$$\text{SIMP-PSD-LEQ}\ \frac{p \leq 0 \qquad PSD(p)}{p = 0}$$

$$\text{SIMP-TSOS-SEQ}\ \frac{p = 0 \qquad TSOS(p, \sum s_i)}{\bigwedge s_i = 0}$$

$$\text{SIMP-TSOS-SNEQ}\ \frac{p \neq 0 \qquad TSOS(p, \sum s_i)}{\bigvee s_i \neq 0}$$

6.4.2.4 Gröbner Bases

There are many ways Gröbner bases can be used during formula simplification[8]. In what follows, let

$$\varphi = \exists \vec{x}(\bigwedge_{i=1}^{k_0} p_i = 0) \wedge (\bigwedge_{i=1}^{k_1} q_i > 0) \wedge (\bigwedge_{i=1}^{k_2} s_i \geq 0) \quad \text{with} \quad p_i, q_i, s_i \in \mathbb{Q}[\vec{x}],$$

with $\mathcal{E} = \{p_1, \ldots, p_{k_0}\}$ the polynomials extracted from the equational fragment of φ.

First, if the equational fragment of a conjunctive formula is unsatisfiable over the complex numbers, then the entire formula is of course unsatisfiable over the real numbers. Thus, checking the triviality of the ideal $\mathcal{I}(\mathcal{E})$ has the potential to detect the unsatisfiability of φ.

Second, a Gröbner basis for $\mathcal{I}(\mathcal{E})$ can be used to inject the polynomials q_i, s_i appearing in *inequalities* into their respective residue classes in the quotient ring $\mathbb{Q}[\vec{x}]/\mathcal{I}(\mathcal{E})$. This process can make nontrivial equalities between different polynomials visible, which can then make it easier for subsequently applied techniques to decide the satisfiability of φ.

Third, the process outlined above can be further extended by splitting a non-strict inequality into its requisite equational and strict inequality components, and examining the resulting subcases. This strengthens the equational fragment (and hence Gröbner reduction) of one subcase, and increases the number of strict inequality atoms in the other. This can be exploited in the context of *full-dimensional cylindrical algebraic decomposition* (FD-CAD). We will examine this in detail in the next section, but let us say a bit about it for now. The key points are as follows.

- CAD can be made much more efficient if the semialgebraic set defined by the formula being analysed is known to be an open set under the Euclidean topology on $\mathbb{R}^n$. If this is known, then a restricted variant, FD-CAD, may be used, which avoids much expensive processing (chiefly, irrational algebraic number computations).

- This openness can be guaranteed if the relation symbols in the formula being analysed are only strict inequalities. That is, FD-CAD can be applied to a formula consisting purely of a boolean combination (in our case, conjunction) of strict polynomial inequalities.

[8]Note that for these simplification techniques based on Gröbner bases, we have opted not to present them in the form of simple inference rules as we have with most others. This is because these rules rely on examining an entire conjunctive formula, not just a single atom within it. We will see in the next chapter how these techniques can be applied in the context of our tool **RAHD**.

- By decomposing non-strict inequalities into their requisite strict inequality and equational cases, one of the two cases of our formula is now closer to being topologically open and thus suitable for FD-CAD, while the other case now has a richer equational structure inducing a potentially larger ideal, which can be exploited by Gröbner basis calculations resulting in more substantial term reductions[9].

Let us illustrate this with an example.

Example 6.4.14. *Let* $\varphi = ((p_1 \leq 0) \wedge \psi)$ *s.t. (WLOG)* ψ *consists only of conjoined strict inequalities and equations. Let* φ *be split into* $\varphi_1 = ((p_1 < 0) \wedge \psi)$ *and* $\varphi_2 = ((p_1 = 0) \wedge \psi)$*, which will both be checked for satisfiability. Observe that the ideal generated by the equations in* φ_2*,* $\mathcal{I}(\mathcal{E}_2)$*, is a (possibly non-strict) superset of the corresponding ideals of* φ *and* φ_1*. Now, fix a monomial ordering* $\prec$ *and reduce all polynomials appearing in the strict inequalities in* φ_2 *with respect to* $GB_{\prec}(\mathcal{I}(\mathcal{E}_2))$ *to obtain an equisatisfiable formula* φ_2'*. Observe that the strict inequalities in* φ_2' *have now been potentially enriched with information contained in the equations of* φ_2*. We can now use the above observation on unsatisfiable subsets of conjoined constraints and examine the satisfiability of only the strict-inequational fragment of* φ_2'*. As this fragment is open, we may now use FD-CAD to decide its satisfiability, and an answer of "UNSAT" would imply the unsatisfiability of the equational branch of* φ*,* φ_2*.*

All of these observations can be further enhanced by saturating the ideal $\mathcal{I}(\mathcal{E})$ by both its (complex) radicalisation and approximations of its real radicalisation (cf. **Section 6.4.1.2**).

Finally, there is much that can be done with both *elimination ideals* and in the special case of zero-dimensional systems. We refer the reader to [Stu02, Dol00] for more on these uses of Gröbner bases during **RCF** decisions.

6.4.2.5 Degree Shifts

The multivariate total degree of polynomials appearing in an $\exists$ **RCF** formula is a key parameter of the asymptotic complexity of most proof methods. Chiefly, if a formula can be made *linear*, then of course much simpler more efficient proof methods may be used, e.g., the simplex algorithm [Nas00]. Moreover, if a formula can be made to be

[9] Observe that super-ideals correspond to sub-varieties, and thus increasing an ideal takes one closer to the empty variety, which is the geometric object corresponding to an unsatisfiable formula.

at most *quadratic*, powerful techniques based upon quadratic virtual term substitution can be used, which often scale to much higher dimensions than procedures based upon cylindrical algebraic decomposition, for instance [Wei97].

Naturally, one way to reduce the total degree of a formula is to eliminate variables which occur in high degree. When that is no longer possible, the following "degree shift" machinery due to Dolzmann [Dol00] can be very useful. A key property of this degree shift is that it allows one to reduce the degree of a variable while not affecting the degrees of any other variables. It can be done for all variables in a formula in turn.

Observation 6.4.15. Let $\varphi = \exists x \psi(x, \vec{y})$ with ψ quantifier-free in negation normal form (i.e., all $\neg$ symbols have been pushed inwards). Let $x^{d_1}, \ldots, x^{d_k}$ be the occurrences of x in the monomials of ψ. Suppose that $d = gcd(d_1, \ldots, d_k) \neq 0$. Let ψ' be the formula obtained from ψ by replacing x^{d_i} by x^{c_i}, where c_i is the cofactor of d and d_i, i.e., $d_i = c_i d$. Then,

$$\exists x\ \psi(x, \vec{y}) \iff \begin{cases} \exists w\ \psi'(w, \vec{y}) & \text{if } d \text{ is odd,} \\ \exists w\ (w \geq 0 \wedge \psi'(w, \vec{y})) & \text{if } d \text{ is even.} \end{cases}$$

This equivalence is easily seen by using $w = x^d$ and $x = \sqrt[d]{w}$ for the $(\Rightarrow)$ and $(\Leftarrow)$ directions, resp.

Example 6.4.16. *Let* $\varphi = \exists x(y_1 x^2 + y_2 < 0)$. *Then,* $d = 2$, *and so we have*

$$\varphi \iff \exists w(w \geq 0 \ \wedge\ y_1 w + y_2 < 0),$$

which can be verified by setting $w = x^2$ $(\Rightarrow)$ *and* $x = \sqrt[2]{w}$ $(\Leftarrow)$, *the latter which must exist by the condition* $w \geq 0$.

6.5 Conclusion

In this chapter, we have presented a large heterogeneous collection of $\exists$ **RCF** proof procedures. In the process, we have paid special attention to how these procedures may be easily combined with each other so as to increase their collective efficacy. At times, this has required us to generalise known procedures and allow other proof procedures to be given to them as functional parameters. Each of these methods have been implemented in our tool **RAHD** which we will see in **Chapter 8**. This will allow us to begin synthesising and experimenting with specific combinations. With this arsenal in hand, we will in the next chapter present the framework of Abstract

Partial CAD. This framework will allow us to use *fast, sound but incomplete* proof procedures built from the components in this chapter, for instance, as parameters for strategically augmenting *complete* CAD-based decision methods.

Chapter 7

Abstract Partial Cylindrical Algebraic Decomposition

7.1 Introduction

In this chapter, we present the framework of *Abstract Partial Cylindrical Algebraic Decomposition* (AP-CAD). This is an extension we have developed of the well-known **RCF** quantifier elimination procedure *partial cylindrical algebraic decomposition* (P-CAD). In this extension, arbitrary (sound but possibly incomplete) $\exists$ **RCF** proof procedures — such as those we have developed in this thesis up to now — can be given as parameters and used to "short-circuit" certain expensive computations during CAD construction. These $\exists$ **RCF** proof procedures may be used to reduce the expense of the stack construction ("lifting") phase of CAD. We will explain this terminology shortly, and will be greatly aided by the fact that restricting ourselves to purely $\exists$ formulas makes CAD much simpler to describe.

Let us spend a moment more on motivation. We are interested in developing feasible proof procedures for $\exists$ **RCF** which can be used for large problems arising in practical verification efforts, specifically those in many variables ("high-dimensional"). Of the complete methods, CAD is the decision method which usually performs best in practice, even though its practical reach is limited to problems in relatively low numbers of variables[1]. Given an **RCF** formula φ, CAD has time complexity doubly exponential in the number of variables of φ, and polynomial in the number of polynomials

[1]For example, we have never succeeded in using pure P-CAD on a nonlinear problem in more than 10 variables. Often, we have observed standard P-CAD implementations such as QEPCAD-B run out of resources on relatively small problems in 5 or 6 variables. QEPCAD-B is a state-of-the-art, careful implementation with many optimisations. The problem is one of inherent complexity of the algorithm.

in φ, the multivariate total degree of the polynomials in φ, the bit width of the coefficients of φ, and the number of atoms of φ. Clearly, high-dimensionality is the most difficult practical hurdle.

Thus, our programme in the previous chapter has been as follows:

1. To investigate a battery of *fast, sound but incomplete* ∃ **RCF** proof methods, each with their own strengths and weaknesses, paying close attention to how these different methods may be compellingly combined. We are especially interested in proof procedures, such as those based on ICP, which scale well to high-dimensional settings.

2. To develop many of these combinations (e.g., Extended Tiwari with ICP, Extended Tiwari with external saturation, ICP with pre-processing based upon degree-shifts, and so on), and to do so in such a way that the exact nature of these combinations can be tailored as needed. For instance, ICP methods of varying strength can be used as the ICP engine in our extended Tiwari calculus. Similarly, the exact saturation methods used during the extended Tiwari method can be instantiated as appropriate for a given application. Formally, this generality is obtained by making key aspects of the combinations *parameters*.

Finally, we will in this chapter present the most general of our combined methods, AP-CAD. This method is especially interesting as, unlike the rest, it is *complete*. Thus, we will build a framework so that *fast, sound but incomplete* methods can be used to improve the processing of this complete method. We will see in the next chapter a simple *proof strategy* language used in our tool **RAHD** for combining ∃ **RCF** proof procedures. Formally, the *proof procedure parameters* used to enhance and tailor AP-CAD as needed will be realised as **RAHD** *proof strategies*.

7.2 CAD Preliminaries

For a detailed overview of CAD, including proofs of the foundational theorems, we refer the reader to [ACM84] and [BPR06]. In what follows, we will present only the background on CAD required to understand AP-CAD for purely ∃ formulas. Our exposition is original and we believe these restrictions on our presentation help make CAD rather more approachable for those interested only in **RCF** satisfiability.

§

CAD is a quantifier elimination algorithm for the full first-order theory of **RCF**. From the outside, it is performing the same task as the Muchnik procedure given in **Chapter 2**, though with a far superior (still hyper-exponential, but elementary) time complexity. Unlike the Muchnik method, CAD is fundamentally geometrical in nature and has an intuitive geometric explanation.

Recall that a *semi-algebraic set* is any subset of $\mathbb{R}^n$ definable by a quantifier-free formula in the language of ordered rings. Then, an *algebraic decomposition* of $\mathbb{R}^n$ is a decomposition of $\mathbb{R}^n$ into finitely many connected components such that each component is semi-algebraic. A *cylindrical algebraic decomposition* is a special type of algebraic decomposition in which the connected components are in a sense "well-behaved" with respect to projections onto lower dimensions.

Before delving into the technical details of this good behaviour, let us first discuss its practical ramifications. From now on, when we say "the polynomials of (an $\exists$ **RCF** formula) φ," we mean the collection of polynomials obtained by zeroing the RHS of every atom in φ through subtracting the RHS from both sides. We assume each such $\exists$ **RCF** formula is in prenex normal form, so that it is a boolean combination of *sign conditions*, i.e., of atoms of the form

$$(p \odot 0) \qquad \text{with} \qquad p \in \mathbb{Z}[\vec{x}], \;\; \odot \in \{<, \leq, =, \geq, >\}.$$

The key point is that if we have in hand a CAD of $\mathbb{R}^n$ derived from an $\exists$ **RCF** formula φ, we can decide the truth of φ from the CAD directly. The reason is amazingly simple: The polynomials of φ will induce a CAD — a decomposition of $\mathbb{R}^n$ into finitely many semi-algebraic connected components or *cells* $c_1, \ldots, c_m \subseteq \mathbb{R}^n$, s.t. every polynomial in φ has *constant sign* on each c_i. This is referred to as the *sign invariance* of a cell decomposition. Given p a polynomial of φ and a c_i a cell, we have

$$\forall \vec{r} \in c_i (p(\vec{r}) = 0) \;\; \vee \;\; \forall \vec{r} \in c_i (p(\vec{r}) > 0) \;\; \vee \;\; \forall \vec{r} \in c_i (p(\vec{r}) < 0).$$

Given this decomposition, it is clear there are only *finitely many* combinations of sign conditions the polynomials of φ may take on over $\mathbb{R}^n$. Thus, to decide φ, we simply substitute a *sample point* from each c_i into the quantifier-free matrix of our formula $QF(\varphi)$ and see if it ever evaluates to **true**. It will evaluate to **true** on at least one sample point if and only if φ is **true** over $\mathbb{R}^n$.

Now, a few questions come to mind after this intuitive description. First, given φ, how does one construct such a CAD? Second, how does one extract these sample points from a description of the cells? Third, given a sample point from each cell, how

does one perform the substitution and formula evaluation when sample points involve irrational algebraic numbers?

We will explain the process of computing CADs for $\exists$ **RCF** formulas. In doing so, we will see that the answer to the second question is given by this CAD construction. When we construct a CAD, what we actually do is construct an *implicit description* of the CAD given by a collection (structured as a tree) of sample points, one for each cell. By sign invariance, one sample point from a cell will be as good as any other. In fact, this will be done by induction so that a sample point in $\mathbb{R}^{i+1}$ will be obtained from a sample point in $\mathbb{R}^i$ by simply appending an additional real number to the vector of real numbers which is the lower-dimensional sample point. This process will result in the construction of a *tree* of cells, with a collection of sample points for each dimension $1 \ldots n$, and a sample point in $\mathbb{R}^i$ giving rise to a collection of sample points in $\mathbb{R}^{i+1}$, each one obtained by appending a different real number to the vector of real numbers which is the sample point in $\mathbb{R}^i$.

The answer to the final question above, regarding the substitution of irrational algebraic numbers, is that one must be careful and use special algorithms for computation with algebraic numbers. This is often a bottleneck of CAD computations. For our purposes below, we will avoid this complication first by working at a higher level of abstraction in which we give ourselves the freedom to speak of manipulating real algebraic numbers by substituting them into $\exists$ **RCF** formulas and evaluating the grounded atoms. This will only be done when this level of abstraction is appropriate for the presentation of our general framework. Later, when we are more concrete and describing our implementation, we will present the "full-dimensional" case of AP-CAD. This full-dimensionality will mean that we can always extract a *rational* sample point from each of our cells, as can be done for $\exists$ **RCF** formulas only involving $\wedge, \vee$ combinations of polynomial *strict* inequalities, for instance. The AP-CAD implementation in our **RAHD** tool is this full-dimensional case.

7.3 CAD Definitions

A CAD of $\mathbb{R}^i$ will be a special type of decomposition of $\mathbb{R}^i$ into finitely many connected components. Let us fix some notation.

Definition 7.3.1 (Region)**.** A region of $\mathbb{R}^i$ is a connected component of $\mathbb{R}^i$.

Definition 7.3.2 (Cell)**.** A *cell* is a *region of a CAD*.

A CAD consists of a collection of cells. We will build the definition of a CAD by induction on dimension.

Our base case is $\mathbb{R}^1 = \mathbb{R}$. A CAD of $\mathbb{R}$ is a decomposition of $\mathbb{R}$ into finitely many cells $c_i \subseteq \mathbb{R}$ s.t. each c_i is of the form

- $\{\alpha\}$ for an algebraic real number α, or
- $]\alpha_1, \alpha_2[$ for algebraic real numbers α_1, α_2, or
- $]\text{-}\infty, \alpha[$ or $]\alpha, +\infty[$ for an algebraic real number α.

Equivalently, a CAD of $\mathbb{R}^1$ is an *algebraic decomposition* of $\mathbb{R}$ into finitely many cells, with each cell either a singleton pointset or an open interval. Given a finite collection of univariate polynomials $P = \{p_1, \ldots, p_k\} \in \mathbb{Z}[x]$, the CAD induced by P is simply the collection of roots of the polynomials p_i and the open intervals induced by these roots (in exactly the same manner we saw within the definition of an *ordered row of signs*, cf. **Definition 2.2.3**).

Example 7.3.3. Let $P = \{x-1, x^2-2\}$. Then, the CAD induced by P is as follows:

$$\begin{aligned} c_1 &=]\text{-}\infty, -\sqrt{2}[, \\ c_2 &= \{-\sqrt{2}\}, \\ c_3 &=]-\sqrt{2}, 1[, \\ c_4 &= \{1\}, \\ c_5 &=]1, \sqrt{2}[, \\ c_6 &= \{\sqrt{2}\}, \\ c_7 &=]\sqrt{2}, +\infty[. \end{aligned}$$

An important observation about a CAD of $\mathbb{R}^1$ is that there is a *natural ordering* between the cells. That is, in the above example it makes sense to say

$$c_1 < c_2 < c_3 < c_4 < c_5 < c_6 < c_7.$$

This ordering property of cells will be fundamental to the definition of CAD in higher dimensions. It is at the heart of the "good behaviour" we mentioned which makes an algebraic decomposition a cylindrical one. This ordering property will allow us to obtain a CAD for $\mathbb{R}^{i+1}$ from a CAD for $\mathbb{R}^i$. Let us see how.

Observe that for CADs of $\mathbb{R}^1$, there are essentially two types of cells — singleton pointsets and open intervals. This dichotomy will continue in higher dimensions with the distinction between *sections* and *sectors*.

In what follows, let $\mathcal{A}$ be a region of $\mathbb{R}^i$.

Definition 7.3.4 (Cylinder)**.** We call $\mathcal{A} \times \mathbb{R}$ the *cylinder over* $\mathcal{A}$ and denote it by $Z(\mathcal{A})$.

Definition 7.3.5 (Stack)**.** Let $f_1, \ldots, f_k \in \mathcal{C}(\mathcal{A}, \mathbb{R})$. That is, f_j is a continuous function from $\mathcal{A}$ to $\mathbb{R}$. Furthermore, suppose that the images of the f_j are ordered over $\mathcal{A}$ in the following sense:

$$\forall \alpha \in \mathcal{A} \left(f_j(\alpha) < f_{j+1}(\alpha) \right).$$

Then, $f_1, \ldots, f_k$ induce a *stack* S over $\mathcal{A}$, where S is a decomposition of the cylinder $Z(\mathcal{A})$ into $2k+1$ regions of the following form:

- $r_1 = \{\langle \alpha, x \rangle \mid \alpha \in \mathcal{A}, x < f_1(\alpha)\}$,
  $r_3 = \{\langle \alpha, x \rangle \mid \alpha \in \mathcal{A}, f_1(\alpha) < x < f_2(\alpha)\}$,
  $\vdots$
  $r_{2k-1} = \{\langle \alpha, x \rangle \mid \alpha \in \mathcal{A}, f_{k-1}(\alpha) < x < f_k(\alpha)\}$,
  $r_{2k+1} = \{\langle \alpha, x \rangle \mid \alpha \in \mathcal{A}, f_k(\alpha) < x\}$,
- $r_2 = \{\langle \alpha, x \rangle \mid \alpha \in \mathcal{A}, x = f_1(\alpha)\}$,
  $\vdots$
  $r_{2k} = \{\langle \alpha, x \rangle \mid \alpha \in \mathcal{A}, x = f_k(\alpha)\}$.

We call regions of the odd index form *sectors* and regions of the even index form *sections*.

Observation 7.3.6. Observe that a sector in a stack over $\mathcal{A}$ is always of the same dimension as the cylinder $Z(\mathcal{A})$. Thus, if r_j is a sector in a stack over $\mathcal{A} \subseteq \mathbb{R}^i$ with $\mathcal{A}$ homeomorphic to $\mathbb{R}^i$, then r_j is homeomorphic to $\mathbb{R}^{i+1}$. We call r_j in this case a *full-dimensional* cell.

Observation 7.3.7. Observe that a *rational point* may always be found inside of a full-dimensional cell.

Observation 7.3.8. Observe how the notion of a *stack* preserves a natural ordering between the regions it induces, as we had for cells in a CAD of $\mathbb{R}^1$:

$$r_1 < r_2 < \ldots < r_{2k+1}.$$

Finally, we may give the inductive step of the CAD definition. The idea is that a CAD of $\mathbb{R}^{i+1}$ will be obtained from a CAD of $\mathbb{R}^i$ by constructing a stack over every cell in the lower-dimensional CAD and unioning the stacks to obtain a set of cells in $\mathbb{R}^{i+1}$.

Definition 7.3.9 (CAD in $\mathbb{R}^{i+1}$)**.** An algebraic decomposition C_{i+1} of $\mathbb{R}^{i+1}$ is a CAD iff C_{i+1} is a *union of stacks*

$$C_{i+1} = \bigcup_{j=1}^{2k+1} w_j,$$

s.t. the stack w_j is constructed over cell c_j in a CAD $C_i = \{c_1, \ldots, c_{2k+1}\}$ of $\mathbb{R}^i$.

With the inductive definition of a CAD given, let us now turn to a fundamental property of the CADs we will construct. This property — a formalisation of the *sign invariance* mentioned in the introduction — will allow us to use CADs to make ∃ **RCF** decisions.

Definition 7.3.10 (P-invariance)**.** Let $P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[x_1, \ldots, x_n]$ and $\mathcal{A}$ be a region of $\mathbb{R}^n$. Then, we say $\mathcal{A}$ is *P-invariant* iff every member of P has constant sign on $\mathcal{A}$. That is given any $p_i \in P$,

$$\forall \vec{r} \in \mathcal{A}(p_i(\vec{r}) = 0) \ \vee \ \forall \vec{r} \in \mathcal{A}(p_i(\vec{r}) > 0) \ \vee \ \forall \vec{r} \in \mathcal{A}(p_i(\vec{r}) < 0).$$

Given a set of regions $\{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$, we say the set is P-invariant iff every region $\mathcal{A}_j$ is P-invariant.

Definition 7.3.11 (P-invariant CAD)**.** Let $P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[x_1, \ldots, x_n]$. Then, $C = \{c_1, \ldots, c_m\}$ is a P-invariant CAD iff C is a CAD of $\mathbb{R}^n$ and C is P-invariant.

Observation 7.3.12. Observe that if $P \subset \mathbb{Z}[x_1, \ldots, x_n]$ is the collection of polynomials in an ∃ **RCF** formula φ, and $C = \{c_1, \ldots, c_m\}$ is a P-invariant CAD of $\mathbb{R}^n$, then we can *decide* the truth of φ over $\mathbb{R}^n$ by selecting a *sample point* from each cell c_i and seeing if the quantifier-free matrix our formula, $QF(\varphi)$, evaluated at the sample point of c_i is **true**. $QF(\varphi)$ will be **true** on *at least one* sample point iff φ is **true** over $\mathbb{R}^n$. This is a direct consequence of the following facts: (i) C covers all of $\mathbb{R}^n$, and (ii) each cell c_i is P-invariant. (Again, the one difficulty is *evaluating* $QF(\varphi)$ at an irrational algebraic sample point. To understand the intuitive idea of this overall decision process, however, let us postpone our worries and for now simply accept that these algebraic number computations can be accomplished algorithmically. Then, the use of CADs to decide ∃ **RCF** sentences is incredibly clear.)

7.4 CAD Construction and Evaluation for ∃ RCF

CAD construction for deciding ∃ **RCF** sentences will take place in three steps: projection, base and lifting (often called "extension" or "stack construction"). To utilise

a constructed CAD to decide an $\exists$ **RCF** sentence, a fourth step, formula evaluation, will be executed. Let us sketch these out and then fill in the relevant details. In what follows, assume that φ is an $\exists$ **RCF** formula and $P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[x_1, \ldots, x_n]$ is the collection of polynomials of φ. We will use the convention that variable x_{i+1} will be projected away before variable x_i.

Projection The *projection phase* will begin with P and iteratively apply a *projection operator* $Proj$ of the form

$$Proj : 2^{\mathbb{Z}[x_1,\ldots,x_{i+1}]} \to 2^{\mathbb{Z}[x_1,\ldots,x_i]}$$

until a set of *univariate* polynomials is obtained over $\mathbb{Z}[x_1]$. This process will consist of levels, one for each dimension, and at each level i we will have what is called a *level-i projection set*[2], P_i. This looks as follows:

$$\begin{aligned}
&P_n = P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[x_1, \ldots, x_n],\\
&P_{n-1} = Proj(P_n) \subset \mathbb{Z}[x_1, \ldots, x_{n-1}],\\
&P_{n-2} = Proj(P_{n-1}) \subset \mathbb{Z}[x_1, \ldots, x_{n-2}],\\
&\vdots\\
&P_2 = Proj(P_3) \subset \mathbb{Z}[x_1, x_2],\\
&P_1 = Proj(P_2) \subset \mathbb{Z}[x_1].
\end{aligned}$$

These level-i projection sets will have a very special property: Namely, it will hold that if we have a P_i-invariant CAD of $\mathbb{R}^i$, then we can use this CAD to construct a P_{i+1}-invariant CAD of $\mathbb{R}^{i+1}$. Thus, to start the motor running, we must construct a P_1-invariant CAD of $\mathbb{R}^1$. This is done in the *base phase*.

Base The *base phase* will consist of computing a P_1-invariant CAD of $\mathbb{R}^1$. As P_1 is univariate, this should be a simple process.

Recall our previous discussion regarding sample points: Namely, when we "construct a CAD" inside of a computer, what we actually do is construct an *implicit description* of the CAD given by a tree of sample points, one for each cell in the CAD. By the sign invariance property of CAD cells w.r.t. the polynomials inducing the CAD, one sample point from each cell will be enough to decide our formulas. So, our task in the base phase reduces to computing a sample point

[2] An interested reader familiar with CAD may notice that we speak only of projection sets, not projection factor sets. This is for simplicity of our exposition. All of the relevant machinery pertaining to projection sets will of course work directly with projection factor sets.

for each cell in a one-dimensional CAD. This can be done by *univariate real root isolation* using Sturm sequences or Bernstein bases, for instance. However, there is one obvious caveat: irrational algebraic numbers.

Recall the distinction between *sections* and *sectors*: In a CAD of $\mathbb{R}^1$, a section is a singleton pointset corresponding to a root of a polynomial, and a sector is an open interval either between consecutive roots or going off from a root to -∞ or +∞. Clearly, a rational number may always be found inside of a sector. But, of course, a root of a univariate polynomial may be irrational, and so a section in a CAD of $\mathbb{R}^1$ may consist of a single irrational algebraic number.

The question then arises in the context of this base phase: How does one represent the sample point for such a section (e.g., the irrational algebraic number which is the only point of the section) inside of a computer? In one word: *carefully*. Precisely because such a sample point is *algebraic*, it can be represented by (i) its minimal polynomial (i.e., the irreducible monic polynomial of which it is a root), and (ii) an open interval with rational endpoints containing the sample point and no other roots of its minimal polynomial. This data uniquely identifies the sample point and provides for it a finite description representable inside of a machine. This description can then be computed with in place of the numbers themselves, e.g., [Rio03].

When we give a concrete presentation of AP-CAD shortly, we will do this for the so-called "full-dimensional" case. This will only require we select sample points from sectors, and these will always be rational numbers. So, we will not need to describe the algebraic number computation methods used for handling sample points taken over sections. By a theorem of McCallum, this will be sufficient for deciding the satisfiability of $\wedge, \vee$ combinations of polynomial *strict* inequalities. Nevertheless, if one "forgets" this complication, the spirit of ∃ CAD, even in its general case, is easily understood. This can be achieved by allowing ourselves the freedom to speak of substituting and evaluating formulas upon arbitrary real algebraic numbers. Let us allow this pedagogically useful expository device for the remainder of this intuitive discussion.

Our mission of the base phase is then to construct a representation of a P_1-invariant CAD of $\mathbb{R}^1$ by giving a sequence of sample points in each cell in the CAD. Let us suppose we have done this (à la **Example 7.3.3**) and our sequence of sample points is $s_1 < s_2 < \ldots < s_{2m+1}$. Now, we can turn to *lifting*.

Lifting The *lifting phase* will take an implicit description of a P_1-invariant CAD of $\mathbb{R}^1$ and progressively transform it into a P_n-invariant CAD of $\mathbb{R}^n$. This is done inductively, by a succession of steps which each take an implicit description of a P_i-invariant CAD of $\mathbb{R}^i$ and "lift" it to an implicit description of a P_{i+1}-invariant CAD of $\mathbb{R}^{i+1}$. Each of these inductive steps will happen by constructing a *stack* over every cell in the CAD of $\mathbb{R}^i$ and extracting sample points from each region in the stack. Amazingly, with the freedom we have given ourselves to speak of algebraic reals, all we need to accomplish lifting is a combination of *substitution* and *univariate real root isolation* (i.e., CAD construction over $\mathbb{R}^1$, which we already know how to do).

Let $C = \{c_1, \ldots, c_m\}$ be the P_i-invariant CAD for $\mathbb{R}^i$ which we will lift to a P_{i+1}-invariant CAD of $\mathbb{R}^{i+1}$. Let $S = \{s_1, \ldots, s_m\}$ be our set of sample points, one from each cell in C. Then, for each cell c_j, we will use the sample point $s_j \in c_j$ to construct a set of sample points in $\mathbb{R}^{i+1}$ corresponding to a *stack over* c_j:

1. As $s_j \in \mathbb{R}^i$, we have that $s_j = \langle r_1, \ldots, r_i \rangle$ for some real numbers $r_1, \ldots, r_i$.
2. The real number components of s_j will then give us values to substitute in for the variables $x_1, \ldots, x_i$ in the level-$(i+1)$ projection set P_{i+1}. By doing this substitution, we will obtain a *univariate* family of polynomials.
3. Let $P_{i+1}[s_j]$ denote $P_i[x_1 \mapsto r_1, x_2 \mapsto r_2, \ldots, x_i \mapsto r_i]$. Then $P_{i+1}[s_j] \subset \mathbb{Z}[x_{i+1}]$ is a univariate family of polynomials.
4. Using the same process as we did in the base phase, compute a $P_{i+1}[s_j]$-invariant CAD of $\mathbb{R}^1$. Let this CAD be represented by a sequence of sample points $t_1 < t_2 < \ldots < t_{2v+1} \in \mathbb{R}$.
5. Then, the *stack over* c_j will be represented by the set of $2v+1$ sample points obtained by appending each t_j to the lower-dimensional sample point s_j. That is, our stack over c_j will be represented by the following sequence of sample points $z_1, \ldots, z_{2v+1}$ in $\mathbb{R}^{i+1}$:

   $z_1 = \langle r_1, \ldots, r_i, t_1 \rangle$,
   $z_2 = \langle r_1, \ldots, r_i, t_2 \rangle$,
   $\vdots$
   $z_{2v+1} = \langle r_1, \ldots, r_i, t_{2v+1} \rangle$.

In the above construction, we call the cell c_j (or the sample point representing it,

s_j) the *parent* of the stack $\{z_1,\ldots,z_{2v+1}\}$. Given a sample point z_v in the stack, we also call c_j (or s_j) the *parent* of z_v. We use the word *child* then in the obvious way. Parenthood will be transitive.

Then, the process of lifting gives rise to a tree: Each cell c_j in the lower-dimensional CAD of $\mathbb{R}^i$ will be parent to a sequence of higher-dimensional children which are points in $\mathbb{R}^{i+1}$, i.e., the sample points drawn from the stack constructed over c_j.

Evaluation Finally, we have the *evaluation phase*. Given a suitably sign invariant CAD induced by an ∃ **RCF** sentence φ with polynomials $P=\{p_1,\ldots,p_k\}\subset\mathbb{Z}[x_1,\ldots,x_n]$, this phase will allow us to use the CAD to decide the truth of φ. This is done by substituting each sample point in the CAD into the quantifier-free matrix of the sentence, and seeing if the sentence ever evaluates to **true**. It will evaluate to **true** at *some* sample point in the CAD iff φ is **true** over the real numbers.

More precisely, let $C=\{c_1,\ldots,c_m\}$ be a P-invariant CAD of $\mathbb{R}^n$ represented by sample points $S=\{s_1,\ldots,s_m\}\subset\mathbb{R}^n$. Then,

$$\langle\mathbb{R},+,*,-,0,1,<\rangle\models\varphi \quad\Longleftrightarrow\quad \bigvee_{\vec{r}\in S} QF(\varphi)[\vec{r}],$$

where $QF(\varphi)[\vec{r}]$ is the quantifier-free matrix of φ evaluated at point $\vec{r}$.

From the above discussion (and again abstracting away from complications regarding irrational algebraic numbers), we can extract a simple algorithm for deciding ∃ **RCF** formulas. At a high level of abstraction, the CAD construction portion of this algorithm can be sketched visually by the following diagrams, the first one illustrating the base phase and the second one illustrating the inductive step of CAD construction.

$$P\subset\mathbb{Z}[x_1]\xrightarrow{CAD_{\mathbb{R}^1}}B\subset 2^{\mathbb{R}}$$

$$\begin{array}{ccc} P_{i+1}\subset\mathbb{Z}[x_1,\ldots,x_{i+1}] & \xrightarrow{Proj_{\mathbb{Z}[x_1,\ldots,x_{i+1}]\mapsto\mathbb{Z}[x_1,\ldots,x_i]}} & P_i\subset\mathbb{Z}[x_1,\ldots,x_i] \\ \Big\downarrow{\scriptstyle CAD_{\mathbb{R}^{i+1}}} & & \Big\downarrow{\scriptstyle CAD_{\mathbb{R}^i}} \\ C_{i+1}\subset 2^{\mathbb{R}^{i+1}} & \xleftarrow{Lift_{\mathbb{R}^i\mapsto\mathbb{R}^{i+1}}} & C_i\subset 2^{\mathbb{R}^i} \end{array}$$

Then, given an $\exists$ **RCF** formula φ with polynomials $P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[x_1, \ldots, x_n]$, we can decide the truth of φ as follows:

1. Construct a P-invariant CAD C of $\mathbb{R}^n$ represented by sample points $S = \{s_1, \ldots, s_m\}$.
2. Evaluate the ground formula

$$\bigvee_{\vec{r} \in S} QF(\varphi)[\vec{r}]$$

and return the result.

7.5 Partial CAD

As it stands, the CAD construction algorithm will build a P-invariant CAD induced by the *polynomials* P of an $\exists$ **RCF** formula φ without paying any attention to the logical content of the formula itself. Besides contributing these polynomials inducing the CAD, the formula does not play a part in the decision process until the *evaluation* phase of the CAD-based decision algorithm. This evaluation does not happen until after the *full* P-invariant CAD has been constructed.

But, when performing lifting, i.e., constructing a stack of regions of $\mathbb{R}^{i+1}$ over a lower-dimensional cell $c_j \subset \mathbb{R}^i$, we may be easily able to see — simply by substitution and evaluation — that the formula $QF(\varphi)$ could *never* be satisfied over c_j.

For instance, let

$$QF(\varphi) = \left((x_4^4 + x_3x_2^3 + 3x_1 > 2x_1^4) \wedge (x_1^2 > x_2 + x_3)\right).$$

Then, if c_j is a cell in a P_3-invariant CAD of $\mathbb{R}^3$ represented by the sample point $s_j = \langle 0, 1, 5 \rangle$, then we can see $QF(\varphi)$ will *never* be satisfied over a cell in a stack which is a child of c_j. Thus, we need not perform lifting over c_j. We can simply throw the cell away and avoid the expensive process of lifting over it entirely.

This is the idea behind *partial CAD* when applied to $\exists$ **RCF** formulas: Before performing lifting over a cell in a CAD of $\mathbb{R}^i$, check to see if there are any atoms in your formula which happen to only involve the variables $x_1, \ldots, x_i$. If this is the case, then perform *partial* evaluation of your formula by evaluating those atoms upon your sample point in $\mathbb{R}^i$, and then use simple propositional reasoning to see if this allows you to deduce the truth of your formula.

This can also allow us to find a *satisfying* assignment for the variables in $QF(\varphi)$ without constructing a whole CAD. For instance, let

$$QF(\varphi) = \left((x_4^4 + x_3x_2^3 + 3x_1 > 2x_1^4) \vee (x_1^2 < x_2)\right).$$

Then, if c_j is a cell in a P_2-invariant CAD of $\mathbb{R}^2$ represented by the sample point $s_j = \langle -1, 2\rangle$, then we can see immediately by substitution that $QF(\varphi)$ is satisfiable over $\mathbb{R}^4$. As a vector of real numbers witnessing this satisfiability, we may return

$$\langle -1, 2, r_3, r_4\rangle \quad \text{where } r_3, r_4 \in \mathbb{R} \text{ are arbitrary reals.}$$

The beautifully simple and powerful idea of partial CAD, due to Collins and Hong [CH91], is the dominant paradigm of modern CAD-based decision methods. It has been implemented as the basis of CAD-based quantifier elimination in both the QEPCAD-B and Redlog programs, and one often sees in the literature practitioners taking it for granted (saying "CAD was applied to decide this formula" when really, the fact that *partial* CAD was applied was absolutely crucial in making the computation terminate in a reasonable amount of time).

7.6 Abstract Partial CAD

The ideas we have developed thus far in this thesis give us a perspective from which we can view partial CAD in a new light. From a high level of abstraction, we can see partial CAD to be normal CAD augmented with three pieces of algorithmic data:

1. A strategy for selecting lower-dimensional cells to use for evaluating lower-dimensional atoms in our input formula,

2. An algorithm which when given a cell c_j will construct a formula $F(c_j)$ which, if it both has a truth value and is decided, can be used to tell (i) if the cell c_j can be thrown away (i.e., $F(c_j)$ is decided to be **false**), or (ii) if a satisfying assignment for our formula can be extracted already from a lower-dimensional cell (i.e., $F(c_j)$ is decided to be **true**),

3. A proof procedure which will be used to decide the formulas $F(c_j)$ generated by the algorithm above.

In fact, in their original paper on partial CAD, Collins and Hong make the point that different cell selection strategies could be used and even implement and experiment

with a number of them[3]. For partial CAD restricted to $\exists$ **RCF**, these three pieces of algorithmic data described above would be:

1. Select cells $c_i \in C$ in some specified enumeration order (specified by s):
   $c_{s(1)}, c_{s(2)}, c_{s(3)}, \ldots$.
2. Given a cell c_j represented by a sample point $s_j = \langle r_1, \ldots, r_i \rangle \in \mathbb{R}^i$, the formula $F(c_j)$ will be constructed from our original $\exists$ **RCF** formula φ by the following process:
   (a) Remove the $\exists$ quantifiers for $x_1, \ldots, x_i$ from φ to obtain a new formula φ'.
   (b) Augment φ' by instantiating x_1 with r_1, x_2 with r_2, and so on to obtain a new formula φ''.
   (c) Evaluate all atoms in φ which are ground to obtain a new formula φ'''.
   (d) Replace all unique non-ground atoms with fresh propositional variables to obtain a new formula $F(c_j)$. If the same non-ground atom appears more than once, each occurrence can be given the same propositional variable.
3. Use a decision procedure for propositional logic to decide the status of $F(c_j)$.

Then, if $F(c_j)$ is **false** (i.e., unsatisfiable), the cell c_j can be abandoned and we need not lift over it. If $F(c_j)$ is **true** (i.e., tautologous), then we can extract a witness to the truth of φ from the sample point s_j. Otherwise, we lift over c_j.

These three pieces of data used in the way prescribed give us the widely-used partial CAD of Collins and Hong. But, from this point of view, we see that there are *many other choices* we could make for these data. Key to this enterprise is that we will be able to pass $\exists$ **RCF** proof procedures as parameters to our AP-CAD procedure. These procedures will be presented in a *proof strategy* language we will introduce in the next chapter in the context of our tool **RAHD**. Because we have built an arsenal of combined (and *combinable*) **RCF** procedures, we will be able to create non-trivial proof strategies and use them to tailor instances of AP-CAD to our needs.

With generality comes freedom. In the context of a decision problem with inherently infeasible time complexity such as $\exists$ **RCF**, the freedom to tailor a decision procedure to one's needs can mean the difference between obtaining a solution in a reasonable amount of time or simply running out of resources (time, space, patience).

[3]For Collins and Hong, a cell selection strategy selects *single* cells in some specified order. In Abstract Partial CAD, cell selection strategies will select *sets* of cells in some specified order and $\exists$ **RCF** proof procedures will be applied to see if every cell in a selected set of cells may be eliminated.

7.6.1 Stages, Theatres and Lifting

Let us now give a formal description of AP-CAD. The framework of AP-CAD will be even more general than the motivating sketch we gave with partial CAD above. This extra generality will be rooted in the fact that a *cell selection strategy* will actually select a *subset* of the cells in the P_i-invariant CAD of $\mathbb{R}^i$. The fundamental notion will be that of an *stage*[4]. In what follows, let $\mathcal{L}_{\exists OR}$ be the fragment of the language of ordered rings consisting of purely $\exists$ sentences in prenex normal form.

While working through the definitions below, it may help the reader to see a concrete instantiation of Abstract Partial CAD. This can be found in the experimental evaluation section of the next chapter, **Section 8.6.2**.

Definition 7.6.1 (Stage)**.** A *stage* will be given by three pieces of algorithmic data. A stage will not formally depend upon the dimension i of the space $\mathbb{R}^i$ decomposed by the CAD C_i. But, for concreteness, we will describe a stage by how it acts in the context of such a fixed (but arbitrary) $\mathbb{R}^i$. That is to say, a stage should be *dimensionally agnostic*: it must satisfy the requirements below for each $i \in \mathbb{N}^+$.

These data are as follows:

1. A *cell selection strategy* for selecting *subsets* of C_i for analysis (we call such a subset a "selection of cells"),

2. A *formula construction strategy* for constructing an $\exists$ **RCF** formula whose truth value will correspond to the relevance of a selection of cells (we call such a formula a "cell selection relevance formula"),

3. An $\exists$ **RCF** proof procedure used to (attempt to) decide the truth or falsity of a cell selection relevance formula.

Let us make these precise. It is important to notice that a cell selection strategy will actually be a strategy for the selection of a set of *sample points*, with each unique sample point drawn from a unique cell. As with CAD generally, this can be seen as a representation of a selection of the corresponding cells. We still call this a *cell selection strategy* as "sample point selection strategy" reads in a misleading manner. A *cell selection function* — the workhorse of a cell selection strategy — will take the set of sample points (from which it will select a subset) as an argument. It will also take a second argument, an integer indicating the "step" of the selection. Given a set

[4]The intended connotation is of a *stage* in a *theatre*.

of sample points, the *covering width function* will indicate how many steps of sample point selection should be executed. This need not result in an exhaustive covering of the set of sample points. For instance, a simple cell selection strategy might, when given the set of sample points $\{s_1, \ldots, s_m\}$, return $\{s_1\}$ for step 1, $\{s_2\}$ for step 2 and so on, with the covering width being m. An equally allowable cell selection strategy would be the same cell selection function with a covering width of 1, resulting in a selection only of $\{s_1\}$.

In the context of CAD construction, sample points will be removed from the set of sample points when they (i.e., their corresponding cells) are deemed to be irrelevant to the $\exists$ **RCF** formula inducing the CAD. This removal might then result in a set of sample points for which the cell selection function behaves differently than it did initially. This motivates the *convergence axiom* for covering width functions, so that these dynamics do not result in a non-terminating CAD-based decision algorithm employing the stage machinery.

Let us abbreviate the set of all finite sets of i-dimensional real vectors (i.e., the set of all possible sets of i-dimensional sample points) as

$$R_i = \{s \subset \mathbb{R}^i \mid |s| < \omega\}.$$

1. A *cell selection strategy* will consist of two components:

   (a) A *covering width function*

   $$w : R_i \to \mathbb{N},$$

   (b) A *cell selection function* (which selects a set of sample points corresponding to a set of cells)

   $$\mathbb{S} : R_i \times \mathbb{N} \to R_i$$

   obeying for all sets of sample points $S_i \in R_i$ and all $j \in \{1, \ldots, w(S_i)\}$ the *containment axiom*:

   $$\mathbb{S}(S_i, j) \subset S_i.$$

   The pair $\langle \mathbb{S}, w \rangle$ will give us an enumeration of a set of subsets of a given set of sample points. We will see how this is used shortly.

2. A *formula construction strategy* will be a function

   $$\mathbb{F} : \mathcal{L}_{\exists OR} \times R_i \to \mathcal{L}_{\exists OR}$$

obeying certain *relevance judgment axioms*. To describe these axioms, we need the context of a fixed (but arbitrary) $\exists$ **RCF** formula and an associated P_i-invariant CAD of $\mathbb{R}^i$.

Let φ be an $\exists$ **RCF** formula with polynomials $P \subset \mathbb{Z}[x_1,\ldots,x_n]$ and let $P_n,\ldots,P_1$ be a sequence of level-$(n,\ldots,1)$ projection sets rooted in P (recall $P_n = P$).

Let $C_i = \{c_1,\ldots,c_m\}$ be a P_i-invariant CAD of $\mathbb{R}^i$ with S_i a set of sample points drawn from a subset of the cells in C_i. If we are given a set of sample points $\{s_{a_1},\ldots s_{a_v}\} \subseteq S_i$, then $\triangle(\{s_{a_1},\ldots s_{a_v}\})$ will denote the set of cells from which the sample points s_{a_j} are drawn.

Then, for each set of sample points S_i and each $j \in \{1,\ldots,w(S_i)\}$ the following *relevance judgment axioms* must hold:

$$\textbf{RCF} \models \neg\mathbb{F}(\varphi,\mathbb{S}(S_i,j)) \quad\Longrightarrow\quad \mathcal{N}(\varphi,\mathbb{S}(S_i,j)),$$

and

$$\textbf{RCF} \models \mathbb{F}(\varphi,\mathbb{S}(S_i,j)) \quad\Longrightarrow\quad \textbf{RCF} \models \varphi,$$

where

(a) $\mathcal{N}(\varphi,\{s_{a_1},\ldots s_{a_v}\})$ means that no child (at any ancestral depth, i.e., in a P_{i+1}-invariant CAD of $\mathbb{R}^{i+1}$, in a P_{i+2}-invariant CAD of $\mathbb{R}^{i+2},\ldots$, in a P_n-invariant CAD of $\mathbb{R}^n$) of any cell in the set $\Delta(\{s_{a_1},\ldots,s_{a_v}\})$ will satisfy $QF(\varphi)$.

3. An $\exists$ **RCF** *proof procedure* will be a function

$$\mathbb{P} : \mathcal{L}_{\exists OR} \to \{\textbf{true},\textbf{false},\textbf{unknown}\} \cup \bigcup_{j\in\mathbb{N}^+} \mathbb{R}^j$$

obeying the *soundness axioms*:

$$\mathbb{P}(\psi) = \textbf{true} \quad\Longrightarrow\quad \textbf{RCF} \models \psi$$

$$\mathbb{P}(\psi) = \textbf{false} \quad\Longrightarrow\quad \textbf{RCF} \models \neg\psi$$

$$\mathbb{P}(\psi) \in \mathbb{R}^j \quad\Longrightarrow\quad \textbf{RCF} \models QF(\psi)[\mathbb{P}(\psi)]$$

for arbitrary $\psi \in \mathcal{L}_{\exists OR}$ and with $QF(\psi)[\mathbb{P}(\psi)]$ in the final axiom being the substitution of the j-vector $\mathbb{P}(\psi)$ (or an arbitrary extension of it to the dimension of the polynomials appearing in ψ) into ψ, resulting in a ground formula. In this case, we call $\mathbb{P}(\psi)$ (or its appropriate extension) a *witness* to the truth of ψ.

We will write $\mathfrak{A} = \langle\langle \mathbb{S}, w\rangle, \mathbb{F}, \mathbb{P}\rangle$ *is a stage* to mean that $\mathbb{S}$ is a cell selection function with w its covering width function, $\mathbb{F}$ is a formula construction strategy, and $\mathbb{P}$ is an $\exists$ **RCF** proof procedure.

We will want to have the freedom to give our AP-CAD algorithm a *sequence* of stages, one for each dimension $1, \ldots, n$. The intuition is as follows:

Stages are introduced so that one can present a *strategy* to an underlying CAD decision algorithm which will prescribe a method for the algorithm to recognise when it can short-circuit certain expensive computations. In particular, stages will be used to either abandon cells and no longer have to lift over them, or to abandon CAD construction altogether if a cell is found whose sample point (or its trivial n-dimensional extension) satisfies our input formula.

If we can abandon a cell at a low-dimension, for instance at the base phase or when beginning to lift over cells of $\mathbb{R}^2$, then this can potentially give us *hyper-exponential* savings later: The number of $i+1$-dimensional cells can be exponentially more than the number of i-dimensional cells. Abandoning a cell at dimension 1 could result in an enormous reduction in the number of cells at dimension 5, for instance.

Thus, it makes sense to arrange stages $\mathfrak{A}_1, \mathfrak{A}_2, \ldots \mathfrak{A}_n$ so that stage $\mathfrak{A}_1$ works *hardest* to make relevance judgments about cells. For if $\mathfrak{A}_1$ causes us to throw away cell $c_j \subset \mathbb{R}^1$, then this could lead to huge savings later. Then, stage $\mathfrak{A}_2$ might still work hard but a bit *less* hard, and so on.

This collection of stages gives rise to the notion of an *n-theatre*. In what follows, let Θ be the set of all stages.

Definition 7.6.2 (Theatre)**.** An *n-theatre* $\mathbb{T}$ will be a function

$$\mathbb{T} : \{1, \ldots, n\} \to \Theta.$$

Stage i in a theatre will be used to make judgments about cells in a P_i-invariant (partial) CAD of $\mathbb{R}^i$ (i.e., at level i). With the notion of an n-theatre in hand, let us describe an augmented CAD-based decision method we will use for deciding $\exists$ **RCF** sentences in the framework of AP-CAD. This will only use the stages in a theatre

during the *lifting* phase. Again, we will speak freely of manipulating real algebraic numbers.

Algorithm 7.6.3 (AP-CAD with Theatrical Lifting)**.** *Suppose we are given an* $\exists$ ***RCF*** *sentence* φ *with polynomials* $P \subset \mathbb{Z}[x_1,\ldots,x_n]$*, together with an n-theatre* $\mathbb{T}$*.*

1. ***Projection*** *Compute a sequence of level-i projection sets as follows:*
   $P_n = P$,
   $P_{n-1} = Proj(P) \subset \mathbb{Z}[x_1,\ldots,x_n]$,
   $P_{n-2} = Proj(P_{n-1}) \subset \mathbb{Z}[x_1,\ldots,x_{n-1}]$,
   $\vdots$
   $P_2 = Proj(P_3) \subset \mathbb{Z}[x_1,x_2]$,
   $P_1 = Proj(P_2) \subset \mathbb{Z}[x_1]$.

2. ***Base*** *Use univariate real root isolation to obtain a* P_1*-invariant CAD of* $\mathbb{R}^1$*,* $C_1 = \{c_1,\ldots,c_{2m+1}\}$ *represented by sample points* $S_1 = \{s_1,\ldots,s_{2m+1}\}$*. Set the current dimension* $i := 1$*.*

3. ***Lifting*** *Let* $\mathbb{T}(i) = \mathfrak{A}_i = \langle\langle \mathbb{S}_i, w_i\rangle, \mathbb{F}_i, \mathbb{P}_i\rangle$ *be the stage for dimension i, and* S_i *the set of sample points for the* P_i*-invariant (partial) CAD of* $\mathbb{R}^i$ *over which we need to lift.*

   (a) *Let* $U := w_i(S_i)$ *and let* $j := 1$*.*

   (b) ***While*** $j \leq U$ ***do***

      i. *Let* $\{s_{a_1},\ldots,s_{a_v}\} := \mathbb{S}_i(S_i, j)$*.*

      ii. *Let* $\chi := \mathbb{P}_i(\mathbb{F}_i(\{s_{a_1},\ldots,s_{a_v}\}))$*.*

      iii. *If* $\chi =$ ***true****, then* ***return true****.*

      iv. *If* $\chi = \langle x_1,\ldots,x_w\rangle \in \mathbb{R}^w$ *for some* $w \leq n$*, then*

         A. *Fix an n-dimensional extension of* χ*, e.g.,*

         $$\vec{r} = \langle x_1,\ldots,x_w,0,\ldots,0\rangle \in \mathbb{R}^n.$$

         B. *Evaluate* $QF(\psi)[\vec{r}]$ *and set* $R \in \{$***true***,***false***$\}$ *to this result.*

         C. *If* $R =$ ***true****, then* ***return*** $\vec{r}$ *as a witness to the truth of* φ*.*

         D. *If* $R =$ ***false****, then* ***return true***[5]*.*

[5]This is perhaps the one counter-intuitive part of the algorithm. Note, however, that this is actually correct: By the combination of the second *relevance judgment axiom* for $\mathbb{F}_i$ and the *soundness axioms* for $\mathbb{P}_i$, the fact that $\mathbf{RCF} \models \mathbb{F}_i(\mathbb{S}_i(S_i, j))$ means that φ is **true**. It's just that the witness $\mathbb{P}_i$ computed for the truth of $\mathbb{F}_i(\mathbb{S}_i(S_i, j))$ might fail to be a witness for φ. In this case, we simply know φ is true without knowing a witness for it.

v. *If* $\chi =$ ***false***, *then set* $S_i' := S_i \setminus \{s_{a_1}, \ldots, s_{a_v}\}$, *else set* $S_i' := S_i$.

vi. *If* $S_i' = \emptyset$ *then* ***return false***.

vii. *If* $S_i' = S_i$ *then set* $j := j+1$.

viii. *If* $S_i' \subset S_i$ *then*

A. *Set* $S_i := S_i'$.

B. *Set* $U := w_i(S_i)$.

C. *Set* $j := 1$.

(c) *Now that the above while loop has finished (without recognising* φ *to be* ***true*** *or* ***false****),* $S_i = \{t_1, \ldots, t_u\}$ *contains sample points corresponding to the cells we have not deemed to be irrelevant. We need to lift over them in order to construct a partial CAD of* $\mathbb{R}^{i+1}$. *We will do this by the following process:*

i. *Let* $S_{i+1} := \emptyset$.

ii. ***For*** j ***from*** 1 ***to*** u ***do***

A. *Substitute the components of* t_j *in for the variables* $x_1, \ldots, x_i$ *in* P_{i+1} *to obtain a univariate family* $P_{i+1}[t_j] \subset \mathbb{Z}[x_{i+1}]$.

B. *Use univariate real root isolation to compute a* $P_{i+1}[t_j]$*-invariant CAD of* $\mathbb{R}^1$, *represented by sample points* K_j.

C. *Set* $S_{i+1} := S_{i+1} \cup K_j$.

(d) *Increase the current dimension by setting* $i := i+1$.

(e) *If* $i = n$ *then lifting is done and we may proceed to the* evaluation phase.

(f) *If* $i < n$ *then we loop and begin the lifting process again, but now with the set of sample points* S_{i+1}.

4. ***Evaluation*** *At this point, we have computed* $S_n \subset \mathbb{R}^n$ *which is a set of n-dimensional sample points sufficient for deciding our formula* φ. *We do so by evaluating the ground formula*

$$\bigvee_{\vec{r} \in S_n} QF(\varphi)[\vec{r}]$$

and returning the result.

Theorem 7.6.4 (Correctness of AP-CAD with Theatrical Lifting)**.** *Let us prove the correctness of* ***Algorithm 7.6.3****. This will be straight-forward given the correctness of the classical CAD-based decision algorithm outlined previously, which we accept as given.*

Proof. There are two essential differences between this AP-CAD algorithm and the classical one. These both take place during lifting. In **Algorithm 7.6.3**, we may

- discard a collection of cells if they are deemed to be irrelevant, and
- quit CAD construction altogether and return either **true** or a witness to the truth of our input formula φ, or return **false** in the case that all cells have been discarded.

If $\{s_{a_1},\dots,s_{a_v}\} \subset \mathbb{R}^i$ is a set of sample points for a P_i-invariant partial CAD of $\mathbb{R}^i$, we will say that $\{s_{a_1},\dots,s_{a_v}\}$ **respects the truth of** φ to mean that there is some n-dimensional child of a sample point in $\{s_{a_1},\dots,s_{a_v}\}$ satisfying $QF(\varphi)$ iff φ is **true**.

We will proceed by induction, assuming that the algorithm has constructed a set of sample points $\{s_{a_1},\dots,s_{a_v}\}$ for a P_i-invariant partial CAD of $\mathbb{R}^i$ which respects the truth of φ. The base case is verified by noting that the *base phase* of the algorithm constructs a full set of sample points for a P_1-invariant CAD of $\mathbb{R}^1$ which trivially respects the truth of φ.

Let us first observe that if we discard a collection of cells because they have been deemed to be irrelevant, then we have not affected the soundness of the decision algorithm.

Cells of a P_i-invariant partial CAD of $\mathbb{R}^i$ will only be deemed to be irrelevant when an *stage* $\mathfrak{A}_i$ indicates this is the case. The key line in the algorithm is 3(b)v, where $\chi = \mathbb{P}_i(\mathbb{F}_i(\varphi,\{s_{a_1},\dots,s_{a_v}\}))$. For this discarding to have occurred, we must have $\chi = \textbf{false}$. By the *second soundness axiom* for $\mathbb{P}_i$, this means

$$\textbf{RCF} \models \neg\mathbb{F}_i(\varphi,\{s_{a_1},\dots,s_{a_v}\}).$$

By the *first relevance judgment axiom* for $\mathbb{F}_i$, this means that $\mathcal{N}(\varphi,\{s_{a_1},\dots,s_{a_v}\})$ must hold. Recall that $\mathcal{N}(\varphi,\{s_{a_1},\dots s_{a_v}\})$ means that no child (at any ancestral depth, i.e., in a P_{i+1}-invariant CAD of $\mathbb{R}^{i+1}$, in a P_{i+2}-invariant CAD of $\mathbb{R}^{i+2},\dots$, in a P_n-invariant CAD of $\mathbb{R}^n$) of any cell in the set $\Delta(\{s_{a_1},\dots,s_{a_v}\})$ of cells corresponding to the sample points $\{s_{a_1},\dots,s_{a_v}\}$ will satisfy $QF(\varphi)$. Thus, by our induction hypothesis, removing the cells from our analysis does not affect the soundness of the decision algorithm. In particular, if we have removed all cells, this means that *no* ancestor of the cells at our current dimension can satisfy $QF(\varphi)$. By our induction hypothesis this means that there exists no n-dimensional real vector satisfying $QF(\varphi)$, and thus φ is **false** as the algorithm will report via line 3(b)vi.

Let us now turn to the second difference: **Algorithm 7.6.3** may quit CAD construction altogether and return either **true** or a witness satisfying $QF(\varphi)$.

In the latter case, a witness is only returned if the algorithm verified, by evaluation, that the witness satisfies $QF(\varphi)$. That this does not affect soundness is apparent.

Let us examine the remaining case, when the algorithm returns simply **true** during lifting. The first place this occurs is on line 3(b)iii. This happens when $\mathbb{P}_i(\mathbb{F}_i(\varphi, \{s_{a_1}, \ldots, s_{a_v}\}))$ is equal to **true**. By the *first soundness axiom* for $\mathbb{P}_i$, this means

$$\mathbf{RCF} \models \mathbb{F}_i(\varphi, \{s_{a_1}, \ldots, s_{a_v}\}).$$

By the *second relevance judgment axiom* for $\mathbb{F}_i$, it then follows that φ is in fact **true** over **RCF** and so the soundness of the algorithm is not affected.

Finally, let us consider the second scenario in which this could occur, line 3(b)ivD. In this case, $\mathbb{P}_i(\mathbb{F}_i(\varphi, \{s_{a_1}, \ldots, s_{a_v}\})) \in \mathbb{R}^j$ for some $j \in \mathbb{N}$. By the *third soundness axiom* for $\mathbb{P}_i$, this means that

$$\mathbf{RCF} \models QF(\mathbb{F}_i(\varphi, \{s_{a_1}, \ldots, s_{a_v}\}))[\mathbb{P}_i(\mathbb{F}_i(\varphi, \{s_{a_1}, \ldots, s_{a_v}\}))].$$

But this implies that

$$\mathbf{RCF} \models \mathbb{F}_i(\varphi, \{s_{a_1}, \ldots, s_{a_v}\})).$$

So, as in the last case, by the *second relevance judgment axiom* for $\mathbb{F}_i$, this means that φ is in fact **true**.

Finally, a word on termination of the while loop (cf. line 3b): Consider a pass of the loop. If any sample points in S_i are discarded, then $|S_i|$ is reduced. If no sample points in S_i are discarded, then U remains constant and j is incremented by 1. Thus, the lexicographic product measure $\mu = \langle |S_i|, U - j + 1 \rangle$ is always decreased along the ordinal ω^2. If ever $|S_i|$ is reduced to 0, then line 3(b)vi guarantees termination. Combining this with the fact that the loop termination condition is $(j > U)$, it follows by the well-foundedness of ω^2 that the loop must terminate.

Thus, by the correctness of the classical CAD-based decision algorithm, it follows by induction that **Algorithm 7.6.3** is sound and terminating. □

7.6.2 Concrete Full-Dimensional AP-CAD

With the framework of Abstract Partial CAD presented, let us now describe in more detail some specifics behind our actual implementation. We will see how it can be used in practice in the context of our tool **RAHD** in the next chapter. Moreover, a reader

may find it helpful to see a concrete instance of the AP-CAD framework. This can be found in the experimental evaluation section of the next chapter, **Section 8.6.2**.

As stated previously, our current implementation in **RAHD** is for the "full-dimensional" variant of AP-CAD. This is the AP-CAD analogue of the "full-dimensional" variant of (partial) CAD, which was introduced by McCallum in [McC93]. The version of full-dimensional (partial) CAD given originally by McCallum varies from standard CAD in that one only selects sample points and lifts over *sectors*. What is useful about this is that these sectors will always be full-dimensional, and will thus be guaranteed to contain a *rational* point. In fact, they will contain infinitely many of them. Thus by restricting oneself to full-dimensional CAD, one can avoid altogether having to do computations with irrational algebraic numbers.

McCallum introduced full-dimensional CAD (what he called "CADMD" or "CAD, Maximal Dimension") as a more efficient method for deciding the satisfiability of systems of polynomial *strict* inequalities. The key theorem in his paper, **Theorem 3.1**, proves the correctness of this restricted variant of CAD for strict polynomial systems. Intuitively, this is believable as a set of solutions in $\mathbb{R}^n$ for a system of strict inequalities over $\mathbb{Z}[x_1,\ldots,x_n]$ will of course always be an open set in the Euclidean topology on $\mathbb{R}^n$. Then, the restriction to *sectors* seems plausible, as it would be puzzling if the rigid nature of a *section* sample point was needed. The proof, though, is nontrivial (invoking a theorem of Baire on nowhere-dense sets). We will use this result freely.

McCallum's theorem tells us the following:

- If φ is an $\wedge,\vee$ combination of polynomial strict inequalities, then we can decide φ using a variant of partial CAD in which we only lift over *sectors*.
- In particular, this means we can select our sample points so that they always are vectors of *rational* numbers.
- Thus, we can decide φ without having to perform any irrational algebraic number computations.

This is very nice as in practice, the immense expense of irrational algebraic number operations are often the bottleneck of CAD computations.

Since the work of McCallum, a number of other researchers have contributed to full-dimensional CAD. Brown has given an enhancement of McCallum's projection operator which, though incorrect for standard CAD, is correct when used in full-dimensional CAD [Bro01]. This Brown-McCallum projection operator is advantageous over classical projection operators as it in practice usually computes much smaller

level-*i* projection sets than other operators for standard CAD. In addition, Strzebonski has also given related projection-oriented enhancements to full-dimensional CAD and has implemented them within the Mathematica kernel [Str00].

These changes to CAD construction when one restricts to full-dimensional CAD can be carried over directly to AP-CAD. By a combination of McCallum's theorem and **Theorem 7.6.4**, it is straight-forward to observe that **Algorithm 7.6.3** is correct for $\wedge, \vee$ systems of polynomial strict inequalities when the enhancements described above are used: Namely, one restricts the algorithm to lift only over sectors and to always select rational sample points within the sectors. Moreover, Brown's correctness theorem for Brown-McCallum projection in the context of full-dimensional CAD means that we may also make use of Brown-McCallum projection for full-dimensional AP-CAD.

Since we no longer need to worry about issues with irrational algebraic numbers, the presentation of **Algorithm 7.6.3** is almost fully concrete. The only remaining pieces are to provide a description of

- the method of real root isolation used to construct full-dimensional CADs of $\mathbb{R}^1$,
- the projection operator, $Proj$.

In **RAHD**, we provide two methods for performing univariate real root isolation: one based on Sturm sequences and Cauchy root bounds which we have implemented, and another based on Bernstein bases which we have incorporated through its implementation in the **SARAG** library in **Maxima** [Car06]. Users are given a choice as to which method they prefer for a given problem. Sturm sequences have pathological numerical and complexity-theoretic properties and thus Bernstein bases are almost unilaterally preferred in the literature. However, let us say in passing that with judicious use of pre-processing (factorisation, square-free parts), caching and structure-sharing, we have found Sturm sequences to work comparably well on many classes of problems.

Finally, let us present the Brown-McCallum projection operator that we use currently in **RAHD**. Compared to other projection operators, it is exceptionally simple. Conveniently, we have already included all of the algebraic operators needed for it in previous chapters of our thesis. There are only three involved in Brown-McCallum projection: leading coefficients, discriminants and resultants.

Definition 7.6.5 (Brown-McCallum projection)**.** Let $P = \{p_1, \ldots, p_k\} \subset \mathbb{Z}[x_1, \ldots, x_n]$. For the algebraic operations below, we will view P as being a set of univariate polynomials in $\mathbb{Z}[x_1, \ldots, x_{n-1}][x_n]$. Then, the Brown-McCallum projection operator

$$Proj_{BM} : 2^{\mathbb{Z}[x_1,\ldots,x_n]} \rightarrow 2^{\mathbb{Z}[x_1,\ldots,x_{n-1}]}$$

is given by

$$Proj_{BM}(P) = LC(P) \cup Res(P) \cup Discr(P),$$

where $LC(P)$ is the set of all leading coefficients of members of P, $Res(P)$ is the set of all resultants taken between non-identical members of P and $Discr(P)$ is the set of all discriminants of members of P.

7.7 Future Work

The framework we have built thus far allows strategic algorithmic data to be used during the *lifting phase* of a CAD-based decision algorithm. It would be very interesting to also work out similar machinery to be used during the *projection phase*. This could perhaps be further specialised to allow for different types of parameters to be given depending upon the actual projection operator used. We have one global idea in this direction which seems general and potentially useful:

If a polynomial p in the level-$(i+1)$ projection set P_{i+1} can be recognised by an $\exists$ **RCF** procedure to be positive or negative definite, then we know that when we are constructing a stack over any cell in $c_i \subset \mathbb{R}^i$ with sample point s_i, the univariate instantiation $p[s_i]$ will never contribute a root to the CAD of $\mathbb{R}^1$ which we use to isolate the $i+1$th components of the the extensions of s_i to $\mathbb{R}^{i+1}$, i.e., p will not contribute anything to the sample points of the stack. This could be the basis for a technique which uses $\exists$ **RCF** proof procedure parameters to discard or ignore polynomials in projection sets. Modern methods for using semidefinite programming to perform sums of squares decompositions of polynomials might be especially useful in this respect [Par03]. As future work, we plan to flesh this idea out and extend the AP-CAD framework to allow for strategic control during projection.

7.8 Conclusion

With the results of this chapter, we now have in hand a general framework for using first-class functional parameters, chiefly a *theatre*, to allow *strategic control* over the

processing of a CAD-based decision algorithm. This is to us very satisfying as it provides a *unifying framework* for combining *fast, sound but possibly incomplete* $\exists$ **RCF** proof procedures and using them to enhance a *complete* decision method without threatening its completeness.

In the next chapter, we will see how this framework can be practically applied in the context of our proof tool **RAHD**.

Chapter 8

Real Algebra in High Dimensions (RAHD) Proof Procedure

8.1 Introduction

In this chapter, we give a user-oriented description of our ∃ **RCF** proof tool **RAHD** (Real Algebra in High Dimensions). The mathematics underlying the system has been given in previous chapters. We encourage readers to refer to **Chapter 6** for information on non-CAD based techniques, **Chapter 7** for Abstract Partial CAD and **Chapter 2** for Gröbner bases and general background on **RCF**, $\mathbf{ACF}_0$ and quantifier elimination.

RAHD is a large system and we will not attempt to describe it fully in this chapter. Instead, we will describe it *enough* so that users may easily get started experimenting with it[1]. Even still, our description of the system is quite involved. We imagine many readers might want to skip directly to the illustration of some actual **RAHD** experiments (cf. **Section 8.6**), referring to the rest of this chapter as necessary.

8.2 RAHD Overview

RAHD is a proof tool for orchestrating and applying a heterogeneous collection of **RCF** proof procedures to decide the satisfiability of nonlinear arithmetical formulas over the real numbers. **RAHD** can be used both interactively and automatically. Its interactive mode is designed both to facilitate a practitioner's analysis of ∃ **RCF** formulas and to provide a platform in which customised ∃ **RCF** proof procedures may be

[1] **RAHD** also has an in-program help system available from its interactive toplevel. User may engage the help system by issuing the command `help` at any **RAHD** prompt.

built and applied. This specification of custom proof procedures is done using a simple *proof strategy* language.

8.2.1 Preliminaries

Let us give some notational and system-oriented preliminaries. Knowledge of these will then be freely assumed.

- **RAHD** accepts as input an implicitly $\exists$-closed boolean combination of rational function equations and inequalities called a *goal*. A *case* is a conjunction of polynomial equations and inequalities. A *goalset* is a collection of *cases* whose disjunction is equisatisfiable with the goal. Every case in a goalset is initially *open*. If a case is proved to be unsatisfiable, it is *closed*. If a case is proved to be satisfiable, it is *satisfied*. If every case in the goalset is closed, then the original formula is proved to be unsatisfiable. If any case is satisfied, then the original formula is proved to be satisfiable.

- Atomic **RAHD** proof methods are embodied in *case manipulation functions* (CMFs). CMFs perform SAT-preserving transformations upon cases. The output of a CMF upon a case may be a *subgoal* which is a goal equisatisfiable with the case. A large collection of native CMFs have been implemented in **RAHD** including those for interval constraint propagation, full-dimensional AP-CAD, the variant of the Tiwari method described in **Section 6.3**, and many others.

- **RAHD** has a proof strategy language for defininining heuristic combinations of CMFs. The strategy language makes it possible to define strategies which apply different proof methods depending upon structural properties of the problem being analysed. Once defined, proof strategies can be used as fully-automatic proof procedures and made accessible both from the interactive toplevel and automatic command-line system interfaces (see below).

- **RAHD** proof strategies are "first-class" objects in the sense that CMFs may take proof strategies as parameters. This is how AP-CAD is implemented as a CMF, for instance.

- **RAHD** has basic machinery for building verified systems of forward-chaining rules (called "verified rulesets"). This allows one to build up "lemma libraries"

and apply them to problems. Rules and rulesets can be defined and verified. Ruleset application is a CMF which takes a ruleset name as a parameter.

- **RAHD** provides a number of pre-defined proof strategies. These are all written in the proof strategy language and may be modified by end-users without touching the **RAHD** source code or rebuilding the system.

- **RAHD** has both interactive toplevel and command-line interfaces. The interactive mode has proof-tree exploration machinery similar to general-purpose proof assistants. This mode has been designed to aid the development of new proof strategies which can then be installed and used from the command-line interface, e.g., in the context of automatic formal verification tool-chains.

- **RAHD** has a plugin infrastructure allowing the easy integration of external tools. In its simplest form, a plugin connects **RAHD** to an external tool by encapsulating each proof procedure present in the external tool as a CMF. Once a plugin has been installed, the CMFs it publishes may then be used in proof strategies.

8.2.2 A Few Quick Examples

Before diving into too many details, let us gain intuition through a few quick examples showing typical uses of the system. In **Figure 8.1**, we show **RAHD** being invoked as a command-line tool, as it might be used within a formal verification tool-chain. From this interface, all defined proof strategies are available to be brought to bear on problem instances. Note that the command-line option "`-i`" invokes the **RAHD** interactive toplevel. We will spend much of the chapter discussing how this toplevel can be used to build new proof strategies. The remaining screen-shots all take place within the interactive toplevel. In **Figure 8.2**, we show **RAHD** finding a satisfying witness to a formula over $\mathbb{R}^{10}$. In **Figure 8.3**, we show **RAHD** proving a formula to be unsatisfiable over $\mathbb{R}^5$. In **Figure 8.4**, we show the help system being invoked to explain the basics of the `check` command which was used in the previous two figures.

8.3 Interactive Toplevel

The **RAHD** interactive toplevel (just "toplevel" for short) provides a shell-like environment for analysing $\exists$ **RCF** formulas and developing and applying proof strategies to decide their status.

```
logicBOX> ./rahd-bin

RAHD: Real Algebra in High Dimensions v0.6a3 (May, 2011)
 designed and programmed by grant o. passmore {g.o.passmore@sms.ed.ac.uk}
  with intellectual contributions from p.b.jackson, b.boyer, g.collins,
  j.harrison, h.hong, f.kirchner, j moore, l.de moura, s.owre, n.shankar,
  a.tiwari, v.weispfenning and many others.  This version is using Maxima
  multivariate factorisation & SARAG subresultant PRS + Bernstein bases.

 Error: Variables and formula not given.

 Usage: ./rahd-bin -v "vars list" -f "formula" <options>
   or   ./rahd-bin [-i | -ip ] [-strategies | -run-strat s | -self-test]

  with options:

    -verbosity q      (0<=q<=10)     degree of proof search output (def: 1)
    -run-strat s                     run defined proof strategy named s
    -strategies                      list all defined proof strategies
    -i                               interactive toplevel
    -ip                              machine-oriented batch evaluator
    -print-model                     print a counter-model, if found
    -print-proof                     print a proof trace, even on failure
    -print-fail                      if a decision is not reached, print
                                      a failure report (unrefuted cases)
    -self-test                       check environment (including plugins)

  where q is rational presented `a/b' or `a' for integers a,b,
        s is a name or ID number of a defined proof strategy.

logicBOX>
```

Figure 8.1: Invoking **RAHD** on the command line. Note that the "`-i`" parameter will load **RAHD**'s interactive toplevel.

```
RAHD: Real Algebra in High Dimensions v0.6a3 (May, 2011)
 designed and programmed by grant o. passmore {g.o.passmore@sms.ed.ac.uk}
  with intellectual contributions from p.b.jackson, b.boyer, g.collins,
  j.harrison, h.hong, f.kirchner, j moore, l.de moura, s.owre, n.shankar,
  a.tiwari, v.weispfenning and many others.  This version is using Maxima
  multivariate factorisation & SARAG subresultant PRS + Bernstein bases.

RAHD!> vars x y z w k1 k2 k3 k4 k5 k6
Current vars: (X Y Z W K1 K2 K3 K4 K5 K6).

RAHD!> assert x >= 12 /\ x > y /\ (x-y) = 1/2 + w
Formula asserted.

RAHD!> assert x > z /\ x^2 + w^3 < (x-y) + k2
Formula asserted.

RAHD!> assert x > k1 /\ k4 = k5^2 /\ k3 > k2 + 2*k4
Formula asserted.

RAHD!> assert k6 <= k4^2 + x*y + 3*k2
Formula asserted.

RAHD!> set print-model
Prover option print-model set.

RAHD!> check
 sat
 model: [X=12,
         Y=11,
         K5=0,
         K1=11,
         K2=1153/8,
         Z=11,
         K3=1161/8,
         K4=0,
         W=1/2,
         K6=4515/8].
```

Figure 8.2: Using **RAHD** to find a satisfying witness to a formula over $\mathbb{R}^{10}$

```
RAHD: Real Algebra in High Dimensions v0.6a3 (May, 2011)
 designed and programmed by grant o. passmore {g.o.passmore@sms.ed.ac.uk}
  with intellectual contributions from p.b.jackson, b.boyer, g.collins,
  j.harrison, h.hong, f.kirchner, j moore, l.de moura, s.owre, n.shankar,
  a.tiwari, v.weispfenning and many others.  This version is using Maxima
  multivariate factorisation & SARAG subresultant PRS + Bernstein bases.

RAHD!> vars a b c d e
Current vars: (A B C D E).

RAHD!> assert a*d + c*b + b*d <= 0
Formula asserted.

RAHD!> assert b >= 0 /\ c >= 0 /\ d >= 0 /\ e >= 0
Formula asserted.

RAHD!> assert a^2 + a*b - b^2 >= 1
Formula asserted.

RAHD!> assert 2*a + b >= 1
Formula asserted.

RAHD!> assert c^2 + c*d - d^2 + 1 <= 0
Formula asserted.

RAHD!> check
 unsat
RAHD:0!u>
```

Figure 8.3: Using **RAHD** to prove a formula unsatisfiable over $\mathbb{R}^5$

```
RAHD!> help check
Usage: check       -or-        check 1.

 Checks the satisfiability of the current context
  using the default proof strategy.

 When a simple "check" is issued, the current assertion
  context is promoted to a goalset and the default strategy
  is used to analyse it.  If any subgoals are generated,
  then the default strategy is applied to them recursively.

 When "check 1" is issued, the process is the same except
  for the following: If subgoals are generated, they will
  be left alone, i.e., no strategy will be automatically
  applied to them.

 To be precise, "check" is effectively the same as

   > build-gs
   > e <default-strategy>,

 while "check 1" is effectively the same as

   > build-gs
   > e1 <default-strategy>.

 * See assert for more on asserting formulas.
 * See default-strategy for setting the default strategy.
 * See show for viewing the current context.
 * See reset for clearing the current context.
```

Figure 8.4: Engaging the help system to learn about the command `check`

When working in the toplevel, **RAHD** maintains data known as the *proof context*. This includes all logical information **RAHD** knows about goals and their goalsets. The **RAHD** *system state* is a larger collection of data which includes the proof context. Data in the system state which is not part of the proof context includes things like cached algebraic computations (e.g., cached Gröbner bases and their reductions, cached factorisations, cached AP-CAD level-i projection sets), and other low-level information. It is intended that users be insulated from these extra-logical things and think of the *proof context* as being **RAHD**'s entire logical world.

Given a formula to analyse, i.e., a toplevel *goal*, one typically works in the following way:

1. *Variables* in the formula are declared.

2. The formula in question is installed as the *toplevel goal* (called `goal 0`) and is made the "active goal."

3. A goalset is built for the toplevel goal, resulting in a number of *open cases*.

4. Through the definition and execution of proof strategies, CMFs are applied — some of which may generate *subgoals* — and users then navigate between the goals (swapping subgoals in as the "active goal"), working on their cases until the system has reached a judgment about the toplevel goal.

8.3.1 Goals and Subgoals

In a given proof context, there is always one *toplevel goal*. This goal — named `goal 0` — is the formula which the system is being used to decide. There may also be *subgoals*. A subgoal is a goal, but only one goal — `goal 0` — is the toplevel goal.

Every goal has an associated *goalset* which is a collection of *cases*, and each case is a conjunction of atoms. The disjunction over the members of the goalset is logically equivalent to the associated goal.

Let G be named `goal X` with goalset $\{c_1, \ldots, c_k\}$. The name `goal X` will carry information as to the placement of G within the proof tree of the toplevel goal. In this way, `X` can be seen as an address. We will make this precise below.

CMFs map cases to equisatisfiable *boolean combinations* of atoms. CMFs may destructively update cases and may also spawn new goals. Goals spawned through the CMF execution are called *subgoals*. When a CMF F maps a case c_i to a new formula

$F(c_i)$ which is purely conjunctive, then the case c_i will be simply replaced with $F(c_i)$ in the goalset of `goal X`. When $F(c_i)$ is not purely conjunctive, then $F(c_i)$ will be turned into a *subgoal* of `goal X`. This subgoal will be named `goal X.i` and will be equisatisfiable with case c_i of `goal X`. (Internally, the implementation of a case also carries around some meta-data, including a history of the CMFs which have modified it, and a pointer to its immediate subgoal, if one exists. We will discuss this more below.)

8.3.2 Toplevel Prompt

When the interactive toplevel is invoked, the user is greeted with a prompt. While working in this toplevel, the prompt conveys key information about the active goal. This information includes the name of the active goal (if subgoals exists), as well as the judgment status of the active goal. The judgment status of the active goal is shown by the final letter of the prompt, if one exists. The following example prompts make the four possibilities clear:

`RAHD!>` — indicating no judgment has been made about the active goal,

`RAHD!m>` — indicating active goal has been judged to be satisfiable and an explicit model has been constructed,

`RAHD!s>` — indicating the active goal has been judged to be satisfiable but no explicit model has been constructed,

`RAHD!u>` — indicating the active goal has been judged to be unsatisfiable.

When more than one goal exists, the name of the active goal is also presented in the prompt. For instance, if our active goal is `goal 0.1.2` and no judgment has been reached about our active goal, then the prompt will be of the following form:

```
RAHD:0.1.2!>
```

Logically, this prompt means that the active goal is equisatisfiable with case 2 of `goal 0.1`, where `goal 0.1` is equisatisfiable with case 1 of the top-level goal, `goal 0`. If we prove `goal 0.1.2` to be unsatisfiable, then this will close case 2 of `goal 0.1`. If we prove `goal 0.1.2` to be *satisfiable*, then this will prove our top-level goal, `goal 0`, to be satisfiable.

8.3.3 Toplevel Commands

We will briefly summarise the commands available at the **RAHD** toplevel. For more detailed information on any command, invoke the in-program help system. For example, to learn more about the `assert` command, issue the command `help assert` at the **RAHD** toplevel. Invoking `help` with no arguments will give a list of all available help topics.

`assert` — Add assertions to the current assertion context. This is done until the assertion context corresponds to the formula whose satisfiability should be checked. All variables in assertions must have been previously declared.

`build-gs` — Build a goalset from the assertion context. This officially promotes the current assertion context to be the toplevel goal and builds a goalset from it.

`check` — Checks the satisfiability of the toplevel goal using the default proof strategy. The default strategy may be changed using the command `default-strategy`. By default, "recursive subgoaling" will be used (see `e` below). To not use recursive subgoaling, invoke `check 1` instead.

`cg` — Change the active goal. This is only applicable when there is more than one goal. Use the command `goals` to see a list of all goals. See the command `cguc` for an often easier way to change to undecided subgoals. Note that `cg` does not propagate judgment status between related goals: For instance, if one is working on a subgoal of a goal and proves to subgoal unsatisfiable, `cg`'ing to the parent goal will not carry the judgment status of the subgoal to the corresponding case in the parent goal. To do this, one must use the command `up`. (This will likely change in future releases.)

`cguc` — Change the active goal to the *n*th undecided (i.e., judgment "unknown") subgoal. When no *n* is given as a parameter, then the active goal is changed to the first undecided subgoal in the ordering reflected by the goal names.

`cmfs` — List all available CMFs.

`default-strategy` — Display and/or re-assign the default proof strategy.

`defrule` — Define a forward-chaining rule.

`defruleset` — Define a ruleset consisting of forward-chaining rules.

`defstrat` — Define a proof strategy.

`e` — Execute an explicitly given proof strategy using recursive subgoaling. This means that if any CMFs applied by the strategy generate subgoals, then the strategy will be applied recursively to the generated subgoals.

`e1` — Execute an explicitly given proof strategy using one-step subgoaling, i.e., without recursive subgoaling. This means that if any CMFs applied by the strategy generate subgoals, then these subgoals will be left alone. It will then be up to the user to `cg` to these subgoals and work on them.

`goal` — See information on the active goal.

`goals` — See a list of all goals.

`goalset` — See a complete list of the status and history of all cases in the active goalset. This is almost always not what a user would want to do, as the output can be astromonical. Instead, the command `opens`, which lists only the open cases in the active goalset, is usually preferable (and its output can often be made much smaller by first applying a few default proof strategies to reduce the number of open cases).

`help` — Invoke the in-program help system.

`lisp` — Execute a raw Lisp form.

`options` — See a list of available prover options. These options can then be modified using `set` and `unset`.

`opens` —See a list of all open cases for the active goal. This includes history information in the form of a CMF trace showing, for each case, the list of CMFs which have progressively modified it.

`pc` — Print a single case (given as an argument the ID number of the case).

`proj-order` — Compute an optimal CAD projection order for the variables in the toplevel goal. This is computed by our implementation of the greedy algorithm put forth by Seidl in his PhD dissertation [DSS04], using the Brown-McCallum projection operator.

`quit` — Cleanly end the **RAHD** session.

`reset` — Reset the current proof context.

`rules` — List all defined forward-chaining rules.

`rulesets` — List all defined forward-chaining rulesets.

`set` — Set a prover option flag.

`set?` — Inspect the value of a prover option flag.

`show` — View the toplevel goal in the original form in which it was installed.

`status` — View the current proof status. This will report the satisfiability status of the toplevel goal, displaying a model if one has been computed. If the active goal is a subgoal, this will also display information on the status of the active goal (e.g., how many open cases remain in the active goal, how many open cases remain in the active goal's parent, if it exists).

`strategies` — List all defined proof strategies.

`strategy` — Display the definition of a particular proof strategy.

`up` — Navigate to the active goal's parent, if it exists.

In the process, any decision reached as to the satisfiability of the active goal will be percolated appropriately to the parent.

In particular, if the active goal is a subgoal and has been found unsatisfiable, the parent's case which generated the subgoal will be closed. If instead the active goal is a subgoal which has been found satisfiable, this implies the satisfiability of the entire parent goal, and this judgment will be inherited by the parent.

`unset` — Unset a prover option flag.

`unwatch` — Stop watching a particular case in the active goalset. (See `watch` below.)

`vars` — Declares variables for use in the proof context. Note that variable names are case insensitive. Variables must be declared before they are used in assertions.

`verbosity` — Set current prover verbosity level to a rational number in the range [0..10]. The default level is 0. The higher the level, the more information is displayed during proof search.

`watch` — Watch a case in the active goalset. This causes the watched case to be printed before every **RAHD** command prompt. This is useful if one is working on a particular case and wishes to observe the changes made to the case by CMF and proof strategy execution.

8.4 Proof Strategies

Heuristic proof procedures are built in **RAHD** through the use of a simple proof strategy language. This strategy language shares much in common with the tactics and tacticals approach of LCF-style interactive proof assistants [Pau87] and the strategy language of PVS [OSRSC99].

The utility of this strategy language is perhaps derived most from the following two properties:

- Proof strategies allow one to conditionally execute different proof procedures based upon structural features of the formula being analysed. This is done through the use of *measure-value conditionals*.

- Proof strategies are first-class objects in the sense that they may be passed around to each other as parameters. This is how Abstract Partial CAD (cf. **Chapter 7**) is realised, for instance.

RAHD ships with a number of predefined proof strategies. If no strategy is defined which is suitable for the class of problems a user is analysing, then one can work to build and define an appropriate strategy. Once this strategy has been defined, it can then be made accessible as an automatic ("push-button") proof procedure. To summarise, **RAHD** has been designed with the following work-flow in mind:

1. Given an $\exists$ **RCF** sentence φ to decide, one first attempts to decide φ through the use of any of the built-in proof strategies.

2. If none of these strategies are able to solve the problem, then the user enters into **RAHD**'s interactive toplevel, installs φ, and analyses the formula interactively.

3. During this analysis, which is done through the manual application of CMFs and execution of strategies, one pays close attention to structural aspects of the problem.

4. Finally, if one is able to decide φ, one can package up the approach used into a defined proof strategy. This strategy can then be made accessible on the command-line and used as an automatic proof procedure.

The grammar of the strategy language is presented in **Figure 8.4.1**. There are currently six primitive measures (cf. grammar class **measure**) on case formulas which are used to build compound measure-value conditionals (cf. grammar classes **value** and **cond**). These six measures are:

`bw` — Sum total bit-width of all rational coefficients of polynomials in case.

`cid` — ID number of the case w.r.t. its containing goalset. This is so that strategies can be targeted to only be applied to specific cases by their ID when needed. This is very useful when proving theorems in the interactive toplevel.

`dim` — Dimension (number of variables) of polynomials in case.

`deg` — Maximal total multivariate degree of polynomials in case.

`gd` — Depth of the goal to which the case belongs (i.e., a case in the goalset of `goal 0` has depth 0, a case in the goalset of `goal 0.2` has depth 1, and so on). This is useful during the execution of strategies using recursive subgoaling.

`nl` — Number of conjuncts in case.

In **Figure 8.4**, we show two simple proof strategies. Note that the second example strategy listed, `stable-simp`, will be used by strategies we define during experiments in **Section 8.6**.

8.4.1 Understanding Strategy Execution

We will informally describe the semantics of strategy execution. We will not be faithful in our description to many aspects of what is actually done in our implementation, as this can be quite convoluted due to efficiency concerns. But, the description we give below is essentially observationally equivalent to the actual mechanism of strategy execution, and it is simple to understand.

To understand strategy execution, one must first understand the only primitive action a strategy performs: the application of CMFs.

Let G be a goal with $\{c_1, \ldots, c_k\}$ the cases in its goalset. Recall that at any given time, a goal has some collection of open cases in its goalset. These are the cases

```
defstrat s-rq-rl-end-no-bg-end
  [interval-cp(max-contractions := 10);
   bounded-gbrni(gb-bound := nl^2, icp-period := 10);
   when (dim <= 5) apcad-fd(stage := div-conq-4);
   split-ineqs;
   simp-zrhs;
   run stable-simp;
   if (dim <= 4) qepcad(open? := 1)
     [if (deg <= 12) redlog-vts
       bounded-gbrni(gb-bound := nl^3, icp-period := 50)];
   interval-cp(max-contractions := 20);
   redlog-vts].


defstrat stable-simp
 [repeat [demod-num; simp-gls; simp-arith]].
```

Figure 8.5: Two example proof strategies

for which we have no satisfiability judgment. Cases may either be marked open (unknown), closed (unsatisfiable) or satisfied (satisfiable). Let $O(G) \subseteq \{c_1, \ldots, c_k\}$ be the collection of open cases for G. Moreover, suppose that no case in $\{c_1, \ldots c_k\}$ has been satisfied. When a CMF F is applied to G, F is mapped over $O(G)$. (Once we see conditional statements in strategies, we will learn that given cases in $O(G)$ may be skipped during the execution of the CMF F because they do not satisfy a specified condition. For now, we ignore this.) For each $c_i \in O(G)$, the following occurs:

1. If $F(c_i)$ is purely conjunctive, then the case c_i is replaced with its image under F, i.e., $c_i := F(c_i)$.

2. If $F(c_i)$ is a non-conjunctive formula, then $F(c_i)$ will be turned into a *subgoal* of G. Recall that every goal has a *name*. For simplicity, let us say G's name is `G`. Then, $F(c_i)$ will be installed as a new goal named `G.i`, and its goalset will be initialised with a collection of cases arising from a DNF normalisation of $F(c_i)$.

3. In both of the above cases, meta-data is recorded as to which CMF manipulated the case c_i, how variables were eliminated (if any were), and so on.

At any time, one can view $O(G)$ as being the "fringe" of a partial proof tree, with CMFs extending the fringe by modifying cases and/or generating subgoals, and at times finding cases to be unsatisfiable (closing a case) or satisfiable (satisfying a case). When $O(G)$ is empty and none of G's cases have been satisfied, G has been proven unsatisfiable. When any of G's cases have been satisfied, G has been proven satisfiable. Let us turn to general proof strategies.

Proof strategies provide a mechanism for conditionally applying CMFs to cases based upon their structural properties. At the core of this conditional processing are case *measures*. These are numerical values which are computed as a function of a case. The primitive measures in **RAHD** are currently `bw`, `cid`, `dim`, `deg`, `gd`, `nl` (see the beginning of this section for a description of their meaning). Measures may be combined to form more numerical values, e.g., `dim^2 + 2*nl`. These polynomials are called *meaure-values*. Finally, conditional statements may be formed based upon boolean combinations of measure-value (in)equalities. These conditional statements act as *guards* which are used to decide whether or not a CMF will be applied to a particular case. These statements are called *measure-value conditionals*.

For example, if the strategy

```
[ when (cid = 15 \/ dim <= 12)
  [apcad-fd(theatre := interval-theatre)] ]
```

is executed in the context of a goal G, then the CMF `apcad-fd` with the parameter given will only be applied to cases which satisfy the guard. For this example, this CMF would only be applied to a case if its ID number was 15 or it was in at most 12 variables.

A natural way to view the function `RUN-STRATEGY` is then as a function of two parameters: A strategy and a guard. When one executes an explicitly given strategy S, one begins the execution with a trivial guard, i.e., one starts by executing the function `RUN-STRATEGY(`S`, true)`. Let us sketch how, at a high-level, this function operates. Note that at any time, **RAHD** has some collection of *defined* proof strategies. These are strategies which have been given a name. If N is the name of a proof strategy, we let $L(N)$ be the strategy named N. We again work in the context of the goal G. An intuitive account of `RUN-STRATEGY` is contained in **Figure 8.4.1**.

Finally, let us discuss what happens if a CMF applied during strategy execution generates a subgoal. **RAHD** has two strategy execution modes. These are called "one-

RUN-STRATEGY(S, $guard$) =

(S is the name of a CMF) $\mapsto$

Apply CMF named S to all cases in $O(G)$ which satisfy the guard $guard$.

(S is [run N]) $\mapsto$

RUN-STRATEGY($L(N)$, $guard$).

(S is [if $cond$ $S1$ $S2$]) $\mapsto$

RUN-STRATEGY($S1$, $guard \wedge cond$) then RUN-STRATEGY($S2$, $guard \wedge \neg cond$).

(S is [when $cond$ $S1$]) $\mapsto$

RUN-STRATEGY($S1$, $guard \wedge cond$).

(S is [repeat $S1$]) $\mapsto$

RUN-STRATEGY($S1$, $guard$) is repeated until no cases in $O(G)$ are modified by its execution.

(S is [$S1$; $S2$]) $\mapsto$

RUN-STRATEGY($S1$, $guard$) then RUN-STRATEGY($S2$, $guard$).

Figure 8.6: An Intuitive Account of the RUN-STRATEGY Function.

step" and "recursive-subgoaling," respectively. It is best to understand this difference in the context of the top-level commands `e1` and `e`. The command `e1` executes an explicitly given strategy using one-step subgoaling, while `e` performs recursive subgoaling.

Imagine entering the following command at the **RAHD** toplevel:

```
> e1 [run waterfall]
```

This command causes strategy execution to work as follows: If a subgoal is generated during the execution of the strategy named `waterfall`, then this subgoal (and its parent case) will be ignored by any subsequent CMFs applied during this execution of the `waterfall` strategy. It will then be up to the user to interactively navigate to this subgoal and execute strategies upon it. However, consider instead the following command:

```
> e [run waterfall]
```

If a subgoal is generated during this execution of the strategy named `waterfall`, then the entire explicitly given proof strategy (i.e., `[run waterfall]`) will be executed upon this subgoal. This happens eagerly, i.e., before the executing strategy moves on to other cases in the active goal's goalset. If any satisfiability judgment is reached about this subgoal, then that judgment will be carried up to the case from which it was generated.

Note that unlike tactics and tacticals in the LCF paradigm, there is no notion of strategies or CMFs failing and throwing exceptions. If a CMF does not make progress on a case, the case is simply left as is. The same holds for the execution of strategies.

Let us carry on now to CMFs in more detail. We will then see in the experimental section (cf. **Section 8.6**) many examples of proof strategies being applied. We encourage the reader to look ahead to that section to help make this exposition more concrete.

8.5 Case Manipulation Functions

The basic inferential mechanism is that of a *case manipulation function* or CMF. The manner in which CMFs are used to modify the proof context has been explained in the previous section. Let us describe the CMFs currently available in **RAHD**.

strategy → action | `if` **cond strategy strategy** | `when` **cond strategy** |
strategy `;` **strategy** | `repeat` **strategy** | [**strategy**] .

action → **cmf-name** | **cmf-name** (**cs-avl**) | `run` **strategy-name** |
`run` **strategy-name** (**cs-avl**) | `print-trace` **int** .

cs-avl → **cs-av** | **cs-av** , **cs-avl** .

cs-av → **cs-arg** := **value** | **cs-arg** := **strategy-name** | **cs-arg** := **theatre-name** .

cond → **a-cond** | **cond** \/ **cond** | **cond** /\ **cond** | **cond** ==> **cond** | ~ **cond** |
(**cond**) | `true` | `false` .

a-cond → **value** = **value** | **value /= value** | **value** != **value** | **value** > **value** |
value >= **value** | **value** < **value** | **value** <= **value** .

value → **measure** | **rational** | **int** | **value** + **value** | **value** - **value** |
value * **value** | **value** ^ **int** | (**value**) .

measure → `bw` | `cid` | `deg` | `dim` | `gd` | `nl` .

Figure 8.7: Grammar of **RAHD** proof strategy language

Tokens of class **cmf-name**, **cs-arg**, **strategy-name** and **theatre-name** are recognised and labelled appropriately by the lexer, as a function of the current system environment (e.g., only strategy names defined prior to parsing time will be recognised to be of class **strategy-name**, and so on). All operations are left-associative except for implication.

8.5.1 Internal CMFs

RAHD has a large number of internal CMFs. These are implemented natively within the **RAHD** system in Common Lisp.

`apcad-fd` — Full-dimensional Abstract Partial CAD (cf. **Chapter 7**) using the Brown-McCallum projection operator. Parameters:

- `stage` — A defined APCAD stage name.
- `theatre` — A defined APCAD theatre name.
- `proj-order-greedy?` — A boolean determining if the Seidl greedy method for determining projection orders should be used (default: false).
- `factor?` — A boolean determining if all polynomials should first be factored before beginning projection (default: true).

`apply-ruleset` — Apply a verified ruleset. Parameters:

- `name` — Name of the verified ruleset.

`bounded-gbrni` — Search for real nullstellensatz witnesses using an extended Tiwari method (cf. **Section 6.3**). Parameters:

- `gb-bound` — A natural number determining an upper-bound on the number of S-polynomials which should be derived before halting search (default: 100).
- `icp-period` — A natural number determining the number of S-polynomials which should be derived in between each execution of ICP upon the growing basis (default: 10).
- `union-case` — A boolean determining whether or not the original case formula should be unioned with the growing basis when ICP is called (default: false).
- `summand-level` — A natural number determining how many levels of sums of the members of the growing basis should be added to the growing basis. This is useful when a real nullstellensatz state witness (cf. **Section 6.3.6**) is not present in the growing basis, but the witness can be obtained simply by summing members of the growing basis. If `summand-level` is L and the

growing basis is $G = \{p_1, \ldots, p_k\}$, then the set of polynomials G^* will be unioned with G at each ICP period, where G^* is defined as follows:

$$G_1 = \left\{ \sum_{p_i, p_j \in G} p_i + p_j \right\},$$
$$G_2 = \left\{ \sum_{p_i, p_j \in G_1} p_i + p_j \right\},$$
$$\vdots$$
$$G_L = \left\{ \sum_{p_i, p_j \in G_{L-1}} p_i + p_j \right\},$$
$$G^* = G_1 \cup \ldots \cup G_L.$$

`saturate-by` — The name of a proof strategy used for saturation during real nullstellensatz search using an extended Tiwari method with ICP (cf. **Section 6.3.3**)). Fresh conjuncts derived by this strategy are added to the growing basis. If this strategy creates subgoals (i.e., a CMF applied by the strategy maps the case to a non-conjunctive equisatisfiable formula), then no conjuncts will be added to the growing basis. Saturation occurs at each ICP period (before the formula is sent to the ICP procedure for analysis).

`canon-tms` — Fully canonicalise all polynomials into sum-of-monomials normal form.

`demod-lin` — Solve for variables appearing linearly in monomials in polynomial equations in the case (cf. **Section 6.4.1.4**) and then perform the derived substitutions. As covered in the referenced section, this is done in a manner which respects the active variable ordering so that a terminating system of variable substitutions is derived.

`demod-num` — Substitute value `v` for variable `x` if `v` is a rational number and the atom `(x = v)` is present in the case.

`factor-sign` — Factor all polynomials appearing in the case and attempt to deduce their sign. If any sign is deduced, the corresponding fact is added as a conjunct to the case. See **Section 6.4.1.5** for details.

`fert-tsos` — Recognise trivial sums of squares and perform simplifications found in **Section 6.4.2.3**.

`full-gbrni` — Perform an unbounded classical Tiwari real nullstellensatz search. As this search is unbounded, it is not usually a good CMF to include in heavily used proof strategies. Nevertheless, it is at times useful as an endgame procedure during problem exploration in the interactive toplevel.

`idm-zpb` — Branch on the nullity of variables if they appear in a zero product. For example, if there is an atom of the form `x*y*z = 0` in the case, then this CMF will map the case to a new formula which is the case conjoined with the disjunction:

```
(x = 0) OR (y = 0) OR (z = 0).
```

`idm-zpb-gen` — The analogue of `idm-zpb` above for arbitrary terms appearing in zero products.

`interval-split` — Split a term at a specified rational value. If term t is split at value v, then a new formula is derived which conjoins the following disjunction with the case:

```
(t < v) OR (t = v) OR (t > v).
```

This causes a subgoal to be generated from the case whose goalset will contain three cases. Parameters:

`tm` The term to split.

`pt` A rational number specifying the point at which the term `tm` should be split.

If no parameters are given, then heuristics are used to select the term and its respective splitting point.

`interval-cp` — Perform interval constraint propagation (cf. **Section 6.2**). Parameters:

`max-contractions` — A natural number specifying the maximum number of contraction steps to execute before returning a computed interval context for the terms appearing in the case.

`quick-sat` — Search for rational satisfying witnesses for the case. This is done by first applying ICP to obtain bounds on each variable in the case, and then by selecting a specified number of sample points inside the containing intervals. When no non-trivial intervals were obtained for a variable, then heuristics are used to select a range of sample points for that variable.

`rcr-ineqs` — Use a Gröbner basis for the ideal $\mathcal{I}$ induced by the equational fragment of a case to inject all polynomials appearing in case inequalities into the residue class ring $\mathbb{Q}[\vec{x}]/\mathcal{I}$. This can be especially useful before invoking an open (full-dimensional) variant of CAD. See **Section 6.4.2.4** for details.

`rcr-svars` — Peform a bounded search for equalities between terms present in the real radical of the ideal induced by the equational fragment of the case. See **Section 6.4.1.2** for details.

`satur-lin` — Derive (ideally, additional) orientations of arithmetical facts so that each variable appearing only linearly in any given atom is made to be the LHS of a derived atom. This is done to increase the contraction efficacy of ICP. See **Section 6.4.1.4** for details.

`simp-arith` — Simplify arithmetical atoms using the "light-weight" arithmetical simplification described in **Section 6.4.2.1**.

`simp-gls` — Simplify ground atoms and terms using the ground rule described in **Section 6.4.2.1**.

`simp-real-null` — Recognise simple real nullstellensatz witnesses, using essentially a combination of state-witness recognition and ICP (cf. **Section 6.3.2.1**).

`simp-zrhs` — Zero the RHS of each atom in the case.

`split-ineqs` — Split atoms in non-strict inequalities into a disjunction of a strict inequality and an equation. For instance, the case

(p >= 0) AND (q >= 0)

is mapped to the equivalent formula

((p > 0) OR (p = 0)) AND ((q > 0) OR (q = 0)).

This results in the generation of a subgoal. Parameters:

`atom` A natural number specifying a specific atom to split (with *non-strict inequality* atoms in a case being labeled from left to right starting at 0). For example, if this parameter is given a value 0, then the first non-strict inequality atom in the case will be split, even though this atom might not be the first atom in the case overall.

`max-splits` A natural number specifying that maximum number of splits to perform. At most the number of splits specified are then performed on the case from left to right.

When no parameters are given, all non-strict inequalities appearing in the case are split.

`triv-ideals` — Compute a Gröbner basis from the equational fragment of the case and check to see if it is equivalent to $\{1\}$. This will recognise if the equational fragment of the case in question is unsatisfiable over the complex numbers. By default, the graded reverse lexicographic ordering (also called the degree reverse lexicographic ordering) is used. This (and the ordering upon the variables inducing the monomial order) can be changed by power users through altering the `*VARS-TABLE*` and `mo<` Lisp parameters. See `polyalg.lisp` and its function `set-active-term-ordering` for how this can be done. See the interactive toplevel command `lisp` for how one executes raw Lisp forms from within a **RAHD** session.

`univ-sturm-ineqs` — Use Sturm sequences to recognise a certain class of unsatisfiable cases. This is done as follows, for each univariate polynomial appearing in an atom (after the RHS of the atom has been made 0) in the case:

- The current interval context is checked to see if rational bounds on the variable of the polynomial are known.
- If rational bounds are known, then a Sturm sequence is computed and used to evaluate the number of roots of the polynomial within the range of the variable.
- If the polynomial has no roots in the range of its variable, then a sample point within the interval for the variable is selected and the polynomial is evaluated upon this sample point.
- Since the polynomial has no roots in the range of its variable, the polynomial only obtains one sign in the context of the case: the sign of the polynomial at the selected sample point.
- Then, this sign (-1, 0 or $+1$) can be substitued for the polynomial itself, and the corresponding ground atom can be evaluated. If it is unsatisfiable, then the case is unsatisfiable.

8.5.2 CMF plugins

RAHD has a plugins mechanism for connecting the system to external tools. A plugin is a bit of Lisp code which acts as an interface between **RAHD** and proof procedures present in the external tool. A plugin creates this connection at the level of a CMF: A plugin may export any number of CMFs, with the intention that each CMF published by a plugin utilise the external tool to do a specific type of real algebraic reasoning. Once plugins have been installed and tested (see the bit about testing below), then they are made available to be used in proof strategies in the same manner as the internal CMFs.

By default, two plugins are installed: One connecting **RAHD** to the partial CAD procedure QEPCAD-B [Bro04] — named `qepcad` — and another connecting **RAHD** to the virtual term substitution procedure found in Reduce/Redlog [AD99] — named `redlog-vts`.

The plugins system has a basic testing framework for verifying that one's environment is working so that the tools linked together by the plugin are operating successfully. This testing framework uses sample $\exists$ **RCF** problems (ideally, a combination of SAT and UNSAT problems) to test the plugin on a small number of instances. Users writing plugins are encouraged to use this simple testing framework. This is especially important if they wish others to be able to make use of their plugin.

Many of the default proof strategies make use of the QEPCAD-B and Reduce/Redlog plugins. If these (or any other) loaded plugins do not pass their tests (i.e., if **RAHD**'s running environment is not setup correctly to make use of the relevant external tools), then the CMF symbols exported by the plugin will be associated with the CMF which is the identity function, i.e., a no-op. This way, proof strategies containing plugin CMFs may always be executed, with applications of CMFs exported by broken plugins simply doing nothing to the proof state. If a user has started an interactive **RAHD** session with broken plugins and has in the mean time fixed their environment, they may ask **RAHD** to recognise this using the toplevel command `refresh-plugins`.

The plugins system has been well-documented in both `plugin.lisp` in the main **RAHD** source code and in `qepcad.lisp` and `redlog.lisp` under the `./plugins/` subdirectory. For this reason, we will say little else about it in our thesis.

8.5.3 Verified Rulesets

RAHD has a simple forward-chaining mechanism allowing users to define, verify and apply their own saturation rules. This is done through the machinery of *verified rulesets*. Let us secure some notation.

A *rule* is a Horn clause. A *ruleset* is a collection of rules. A *verified rule* is a rule which has been proven to be sound. A *verified ruleset* is a collection of verified rules. Rules are defined globally and a rule may be a member of many rulesets.

The full power of **RAHD** may be brought to bear on the verification of rules. A rule gives rise to a *rule goal* which is a conjunction consisting of the hypotheses of the rule and the negation of its conclusion. To verify a rule, its *rule goal* is installed as a goal and must be proved unsatisfiable. Once a rule has been verified, it may be *applied* using the `apply-rule` CMF. It is often convenient to combine multiple rules into a ruleset and apply a verified ruleset to a case. **RAHD** provides the `apply-ruleset` CMF to facilitate this.

Rules are *applied* in the context of a case. If a rule is applied to a case, a matching algorithm is used to see if each of the hypotheses of the rule can be discharged by explicit atoms in the case. This matching includes some simple generalisations (e.g., if the hypothesis is $(x \geq y)$ and the case contains either $(x > y)$ or $(x = y)$ then the matching will succeed). But, this simple matching — which includes polynomial canonicalisation but not associative-commutative ("AC") matching — is the only reasoning used currently for validating rule instantiations. When a valid instance of a rule is found, the instantiated conclusion is added as a conjunct to the current case. We then say the rule application *succeeded*. If it did not succeed, it *failed*.

Rules in a ruleset have an order given by the order they are listed in the definition of the ruleset. Within a ruleset, rules can be made *active* or *inactive*. This status is local to a particular ruleset: A given rule may be active in one ruleset and inactive in another. When one applies a verified ruleset, only the *active* rules are applied. When one is applying a verified ruleset to a case, each active rule is applied in the order given. The success of a rule in a ruleset can influence the success of later rules in the ruleset: If the application of rule R_i is successful, resulting in the current case being extended with new conjuncts, then this extended case is what is used for the application of rule R_{i+1}. For example, the pre-defined verified ruleset `force-sign` is as follows:

```
defrule force-sign-i
 [[Y > 0] /\ [W > 0] /\ [X*Y > 0] /\ [X*Y + Z*W = 0]]
```

```
   ==> [X*Z < 0].

defrule force-sign-ii
 [[Y > 0] /\ [W > 0] /\ [X*Y < 0] /\ [X*Y + Z*W = 0]]
   ==> [X*Z < 0].

defrule force-sign-iii
 [[X > 0] /\ [Y > 0] /\ [X*Y - Z < 0]] ==> [Z > 0].

defrule force-sign-iv
 [[X > 0] /\ [X*Z + W < 0] /\ [Y - X*X = 0]]
   ==> [Y*Z + X*W < 0].

defruleset force-sign
  {force-sign-i,
   force-sign-ii,
   force-sign-iii,
   force-sign-iv}.
```

8.6 Experiments

In this section, we present some experimental results obtained with **RAHD**.

First, we present the development and experimental results of a strategy based on a combination of Gröbner basis computation and full-dimensional partial CAD. This is a concrete instance of the methodology we propose for **RAHD** use: the development of custom automatic proof strategies — combining many $\exists$ **RCF** (semi-)decision methods — through the interactive solving of specific (previously out-of-reach) problem instances, and the subsequent application of these custom proof procedures to other problems (ideally, those with "similar structure"). An earlier and much less comprehensive version of this class of experimental results was published in [PJ09].

Finally, we turn to our proof method of Abstract Partial CAD (cf. **Chapter 7**). We construct an AP-CAD theatre based on generalised interval arithmetic and illustrate its use in detail upon an example $\exists$ **RCF** formula. We contrast the processing of this AP-CAD decision algorithm with that of classical CAD and partial CAD algorithms, observing how they differ during the lifting or stack construction phase. Then, we

perform a similar lifting comparison analysis on four other $\exists$ **RCF** formulas.

8.6.1 A Strategy with Gröbner Bases and Full-dimensional CAD

This class of experiments begins with the following $\forall$ **RCF** formula which was sent to us by John Harrison in 2008:

$$\begin{aligned}
&\forall a\ \forall b\ \forall c\ \forall d \\
&((0 \leq a)\ \wedge\ (a \leq 1)\ \wedge\ (0 \leq b)\ \wedge\ (b \leq 1)\ \wedge \\
&(0 \leq c)\ \wedge\ (c \leq 1)\ \wedge\ (0 \leq d)\ \wedge\ (d \leq 1)) \\
&\Rightarrow \\
&(((1-a^2b^2)(1-cd)(ad-bc)(ad-bc)+ \\
&(2ab)(cd-ab)(1-ab)(c-d)(c-d)+ \\
&(a^2b^2-c^2d^2)(1-cd)(a-b)(a-b))\ \geq\ 0)
\end{aligned}$$

Call the above formula φ. Harrison had attempted without success to prove φ using the powerful `REAL_SOS` tactic in the proof assistant HOL-Light which uses semidefinite programming to search for Positivstellensatz witnesses [Har07]. We then tried, also in vain, to prove it using four additional state-of-the-art methods:

- the partial CAD procedure QEPCAD-B [Bro04],
- the virtual term substitution procedure Reduce/Redlog `rlqe` [AD99],
- the partial CAD procedure Reduce/Redlog `rlcad` [AD99], and
- the branch-and-bound based interval analysis system Realpaver [GB06].

Motivated by the intuition that this formula might be "just out of reach" of CAD, we focused upon ways we might transform φ into a form suitable for a CAD-based decision.

8.6.1.1 A First Approach: Splitting, Simplification, and (FD-)CAD

As we know, the complexity of CAD is dependent most upon the number of variables in the input formula. Thus, techniques which reduce the number of variables in the formula have the potential to be very helpful. We will turn to these shorly. First, let us

switch to proving φ by refuting $\neg\varphi$, so that we work within $\exists$ **RCF**. That is, to prove φ we must refute ψ where

$$\psi = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0 \le a)\ \wedge\ (a \le 1)\ \wedge\ (0 \le b)\ \wedge\ (b \le 1)\ \wedge \\ (0 \le c)\ \wedge\ (c \le 1)\ \wedge\ (0 \le d)\ \wedge\ (d \le 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc)+ \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d)+ \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b))\ <\ 0) \end{array}\right].$$

In examining ψ, we see no immediate way to eliminate any variables. There is, however, one structural property of ψ which is interesting in relation to variable elimination: each variable is explicitly bounded within a compact interval given by intervals with explicit rational endpoints. This bounding is done through the use of two atoms for each variable, the first expressing a lower-bound and the latter an upper-bound, e.g.,

$$(0 \le a)\ \wedge\ (a \le 1).$$

Observe that if we split any of these bounding atoms — take $(0 \le a)$, for instance — into an equivalent disjunction of an equation and a strict inequality, then we may obtain two sub-problems $\psi_{a,=}$ and $\psi_{a,<}$ s.t.

$$\varphi \iff \neg\psi \iff \neg(\psi_{a,=}\ \vee\ \psi_{a,<})$$

where

$$\psi_{a,=} = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0 = a)\ \wedge\ (a \le 1)\ \wedge\ (0 \le b)\ \wedge\ (b \le 1)\ \wedge \\ (0 \le c)\ \wedge\ (c \le 1)\ \wedge\ (0 \le d)\ \wedge\ (d \le 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc)+ \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d)+ \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b))\ <\ 0) \end{array}\right]$$

and

$$\psi_{a,<} = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0 < a)\ \wedge\ (a \leq 1)\ \wedge\ (0 \leq b)\ \wedge\ (b \leq 1)\ \wedge \\ (0 \leq c)\ \wedge\ (c \leq 1)\ \wedge\ (0 \leq d)\ \wedge\ (d \leq 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc)\ + \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d)\ + \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b))\ <\ 0) \end{array}\right].$$

Now, in the first sub-problem, a may be eliminated through the substitution of 0. If we do this followed by trivial algebraic simplification, we obtain the formula $\psi^*_{a,=}$ (equivalent to $\psi_{a,=}$):

$$\psi^*_{a,=} = \left[\begin{array}{r} \exists b\ \exists c\ \exists d \\ ((0 \leq b)\ \wedge\ (b \leq 1)\ \wedge \\ (0 \leq c)\ \wedge\ (c \leq 1)\ \wedge\ (0 \leq d)\ \wedge\ (d \leq 1)) \\ \wedge \\ (((1-cd)(bc)^2\ + \\ (0-c^2d^2)(1-cd)b^2)\ <\ 0) \end{array}\right].$$

So, we now have

$$\varphi \iff \neg\psi \iff \neg(\psi^*_{a,=}\ \vee\ \psi_{a,<})$$

with $\psi^*_{a,=}$ in only three variables and with significantly simpler polynomials than φ. In fact, $\psi^*_{a,=}$ is so much simpler that it can be easily refuted automatically using normal partial CAD, e.g., through the use of QEPCAD-B. Therefore, we have reduced the proof of φ to the refutation of $\psi_{a,<}$. How shall we refute $\psi_{a,<}$?

Given what we know about CAD (cf. **Chapter 7**), it is obvious that constructing a CAD to decide $\psi_{a,<}$ is no easier than constructing a CAD to decide ψ (or indeed φ). The reason is simple: The polynomials inducing the CAD will be the same for all three of these formulas.

Digging deeper into CAD, however, we can make some progress. Recall McCallum's Theorem on full-dimensional cell decompositions (cf. **Section 7.6.2**). One practical byproduct of this theorem is that CAD for $\exists$ **RCF** can be made substantially more efficient if its input formula consists only of conjunctions of *strict* inequalities. When this is the case, one has the following two advantages:

- A much simpler projection operator can be used (e.g., the Brown-McCallum operator (cf. **Section 7.6.2**)),

- Irrational algebraic number computations can be avoided during lifting as one only has to select sample points from full-dimensional cells, i.e., *sectors*, and these sample points may always be made to be rational numbers.

Both of these savings can lead to tremendous improvements in decision feasibility. Thus, we see that $\psi_{a,<}$ is in a precise sense closer to being able to take advantage of McCallum's Theorem than ψ, as the only difference between the two formulas is that there is one less non-strict inequality atom in $\psi_{a,<}$ than ψ.

Now, the next step is obvious, but it is perhaps surprising that it works in practice: We can simply iterate the reasoning outlined above and arrive at a sequence of sub-problems s.t. each sub-problem is either in less variables or in less non-strict inequalities (both as compared to ψ). This will result in reducing ψ to nine sub-problems: eight will be 3-dimensional, one will be 4-dimensional. Six of the eight 3-dimensional problems will contain a non-strict inequality and will be decided using normal partial CAD. Two of the eight 3-dimensional problems will contain only strict inequalities and will be decided using full-dimensional partial CAD. Finally, the single 4-dimensional problem will be decided by full-dimensional partial CAD since it will be a conjunction of only strict inequalities.

To gain more familiarity with the system, let us walk through how this proof can be carried out in **RAHD**. We will first show how to do this proof "manually" so that the user controls each of these steps of splitting inequalities, substituting and simplifying, and applying normal and full-dimensional partial CAD to the resulting sub-problems. Once we show how it can be done manually, we will then show how one can obtain essentially the same proof automatically through the use of a recursively executed proof strategy.

We begin the **RAHD** images after a formula corresponding to ψ has been installed as the toplevel goal and a goalset has been built for it. We use five figures to illustrate the manual proof: **Figures 8.8, 8.9, 8.10, 8.11, 8.12**.

After examining the manual proof, especially the actions executed in **Figure 8.12**, we see there is much repetition in the structure of the proof. Note that we used the `e1` command to execute the strategies in our manual proof. This command executes a strategy only once and does not apply it recursively to any generated subgoals.

Since this proof is so highly structured, we can easily write a short proof strategy to

```
RAHD!> opens

 Printing all of the open 1 cases for goal 0.

 -------     ------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------

       0     ((<= 0 A) (<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D)
              (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0))    (UNKNOWN)


      1 case in goalset (goal 0) awaiting refutation.

RAHD!>
```

Figure 8.8: Partial illustration of “manual” refutation of ψ (1 of 5) : Before this image was taken, ψ was installed as the toplevel goal, i.e., `goal 0`. Then, we display the open cases in the goalset of `goal 0` by `opens`. There is a single case, `case 0`, i.e., ψ.

```
RAHD!> e1 [split-ineqs(atom := 0)]
RAHD:0!> opens

 Printing all of the open 1 cases for goal 0.

 -------     ------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------

       0     ((<= 0 A) (<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D)
              (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0))    ((UNKNOWN-WITH-SPAWNED-SUBGOAL (0 0))
                       (SUBGOAL-SPAWNED-FROM SPLIT-INEQS THEN NIL))


      1 case in goalset (goal 0) awaiting refutation.

RAHD:0!> cguc
RAHD:0.0!>
```

Figure 8.9: Partial illustration of “manual” refutation of ψ (2 of 5) : The splitting of the first non-strict inequality in `goal 0` (by `split-ineqs(atom := 0)`), the display of its updated status (`UNKNOWN-WITH-SPAWNED-SUBGOAL`, displayed by executing `opens`), followed by the changing of the active goal to the generated subgoal `goal 0.0` (by `cguc`).

```
RAHD:0.0!> opens

 Printing all of the open 2 cases for goal 0.0.

 -------     ------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------

       0     ((<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0)
              (= 0 A))    (UNKNOWN)

       1     ((<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0)
              (< 0 A))    (UNKNOWN)


     2 cases in goalset (goal 0.0) awaiting refutation.

RAHD:0.0!>
```

Figure 8.10: Partial illustration of “manual” refutation of ψ (3 of 5) : The display of the two open cases in the goalset for `goal 0.0` (using `opens`). Observe how `case 0` has `(= 0 A)` and `case 1` has `(< 0 A)`.

```
RAHD:0.0!> e1 [when (cid = 0) [demod-lin; run stable-simp]]
RAHD:0.0!> pc 0

 Printing case 0 for goal 0.0.

 -------     ------------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------------

       0     ((<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+ (* (- 1 (* C D)) (* (* B C) (* B C)))
                  (* (* (- 0 (* (* C C) (* D D))) (- 1 (* C D))) (* B B)))
               0))    (UNKNOWN SIMP-ARITH SIMP-GLS DEMOD-LIN)


      2 cases in goalset (goal 0.0) awaiting refutation.

RAHD:0.0!> e1 [when (cid = 0) [qepcad]]
RAHD:0.0!> opens

 Printing all of the open 1 cases for goal 0.0.

 -------     ------------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------------

       1     ((<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0)
              (< 0 A))    (UNKNOWN)


      1 case in goalset (goal 0.0) awaiting refutation.

RAHD:0.0!>
```

Figure 8.11: Partial illustration of "manual" refutation of ψ (4 of 5) : First, the substitution of `0` for `A` and subsequent simplification of `case 0` of `goal 0.0` is done by `[demod-lin; run stable-simp]`. We display the resulting simplified `case 0` by `pc 0`. Note that this is equivalent to our formula $\psi^*_{a,=}$ discussed previously and is only 3-dimensional. Since it has been simplified, we are able to run normal partial CAD directly upon `case 0` and refute it. Then, we show that `case 1` is the only open case remaining in the goalset of `goal 0.0` by `opens`.

```
RAHD:0.0!> e1 [split-ineqs(atom := 0)]
RAHD:0.0!> cguc
RAHD:0.0.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1!> cguc
RAHD:0.0.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1!> cguc
RAHD:0.0.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1!> cguc
RAHD:0.0.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1.1!> cguc
RAHD:0.0.1.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1.1.1!> cguc
RAHD:0.0.1.1.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad(open? := 1)]]
RAHD:0.0.1.1.1.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1.1.1.1!> cguc
RAHD:0.0.1.1.1.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad(open? := 1)]]
RAHD:0.0.1.1.1.1.1.1.1!> e1 [qepcad(open? := 1)]
RAHD:0.0.1.1.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.1.1.

RAHD:0.0.1.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.1.

RAHD:0.0.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.

RAHD:0.0.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.

RAHD:0.0.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.

RAHD:0.0.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.

RAHD:0.0.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.

RAHD:0.0!u> up
Trickled refutation up to Case 0 of Goal 0.

RAHD:0!u> status

 Goalkey: 0,
 Unknown cases: 0 of 1.

 Decision: unsat.
```

Figure 8.12: Partial illustration of “manual” refutation of ψ (5 of 5) : Finishing the proof. Notice how much repetition there is. We will soon see how this may be automated with the use of a proof strategy executed recursively upon subgoals.

accomplish essentially the same proof through the recursive execution of the strategy upon its generated subgoals. We show how to do this in **Figure 8.6.1.1**.

With this final recursive proof strategy, we have arrived at an elegant solution to the problem of deciding ψ. But, what about related problems with similar structure? For instance, what about problems of similar structure in 5 or more dimensions, or those with more complex equational structure than ψ? Will this strategy work for them? This line of questioning will lead us to develop a more intricate approach.

8.6.1.2 A Further Approach: Eager Splitting and Gröbner Bases

Our presented solution for ψ depended upon a few key properties of ψ:

- Every variable in ψ is bound within a compact interval with explicitly given rational endpoints.
- So, splitting any non-strict inequality results in two branches, one in which a variable can be eliminated, and the other in which we are closer to being able to decide the formula using only full-dimensional partial CAD.
- ψ itself is only 4-dimensional, so that once a variable has been eliminated from the equational branches of split inequalities, then the resulting formulas sent to normal partial CAD are only in 3-dimensions.

When confronted with a problem of similar structure in higher dimensions, we can easily run into the problem that even the equational branches in which a variable has been eliminated are still out of reach of normal CAD. And if we consider formulas in which *some* but not all of the variables are bound within explicitly given compact intervals with rational endpoints, then variable elimination need not be so simple.

Motivated by wanting to extend this simple strategy which was succesful on ψ to more classes of problems, let us recall a sketch of a proof strategy we gave in **Section 6.4.2.4** during a discussion of uses of Gröbner bases during $\exists$ **RCF** decisions:

> In what follows, let
>
> $$\varphi = \exists \vec{x}(\bigwedge_{i=1}^{k_0} p_i = 0) \wedge (\bigwedge_{i=1}^{k_1} q_i > 0) \wedge (\bigwedge_{i=1}^{k_2} s_i \geq 0) \quad \text{with} \quad p_i, q_i, s_i \in \mathbb{Q}[\vec{x}],$$
>
> with $\mathcal{E} = \{p_1, \ldots, p_{k_0}\}$ the polynomials extracted from the equational fragment of φ.

```
RAHD!> opens

 Printing all of the open 1 cases for goal 0.

 -------     ------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------

       0     ((<= 0 A) (<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D)
              (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0))    (UNKNOWN)


      1 case in goalset (goal 0) awaiting refutation.

RAHD!> e [demod-lin; run stable-simp;
           [if (dim <= 3) [qepcad]
              [split-ineqs(atom := 0); qepcad(open? := 1)]]]
RAHD:0!u> status

 Goalkey: 0,
 Unknown cases: 0 of 1.

 Decision: unsat.

RAHD:0!u> goals

All goals:
 0.0.1.1.1.1.1.1.1
 0.0.1.1.1.1.1.1
 0.0.1.1.1.1.1
 0.0.1.1.1.1
 0.0.1.1.1
 0.0.1.1
 0.0.1
 0.0
 0

RAHD:0!u>
```

Figure 8.13: Proof strategy used to obtain automatic refutation of ψ by its execution using recursive subgoaling (via `e`). After obtaining the proof, we use the `goals` command to see a list of all (sub)goals which were constructed: Indeed, this automatic proof has the same structure as our manual one. Further, if we navigate down to each subgoal and view the list of CMFs which manipulated each case, we will see that the constructed proof is exactly the same as our manual one except for the fact that the automatic proof performs eight calls of regular CAD (upon 3-dimensional formulas) instead of six. We can make these proofs exactly the same with a slightly more complicated strategy.

> First, if the equational fragment of a conjunctive formula is unsatisfiable over the complex numbers, then the entire formula is of course unsatisfiable over the real numbers. Thus, checking the triviality of the ideal $\mathcal{I}(\mathcal{E})$ has the potential to detect the unsatisfiability of φ.
>
> Second, a Gröbner basis for $\mathcal{I}(\mathcal{E})$ can be used to inject the polynomials q_i, s_i appearing in *inequalities* into their respective residue classes in the quotient ring $\mathbb{Q}[\vec{x}]/\mathcal{I}(\mathcal{E})$. This process can make nontrivial equalities between different polynomials visible, which can then make it easier for subsequently applied techniques to decide the satisfiability of φ.
>
> Third, the process outlined above can be further extended by splitting a non-strict inequality into its requisite equational and strict inequality components, and examining the resulting subcases. This strengthens the equational fragment (and hence Gröbner reduction) of one subcase, and increases the number of strict inequality atoms in the other. This can be exploited in the context of *full-dimensional cylindrical algebraic decomposition* [...]

Let us now elaborate upon this idea. Imagine we are given an $\exists$ **RCF** formula χ. Then, we might consider the following high-level approach to deciding χ:

- First, apply simple and cheap techniques to see if χ can be recognised to be (in)consistent very easily. If not, then continue.

- Assuming that χ does not contain too many non-strict inequalities so as to make the next steps combinatorally infeasible, replace each non-strict inequality $(x \leq y)$ in χ with the disjunction $(x = y \vee x < y)$. Call the resulting formula χ_*. Then, convert χ_* to DNF. Call the resulting formula $DNF(\chi_*)$ with

$$DNF(\chi_*) = \bigvee_{i=1}^{k} C_i.$$

  Let us consider the set $S = \{C_1, \ldots, C_k\}$. We will work to refute χ by refuting every member of S.

- For each C_i which contains equations, perform the "obvious" substitutions and algebraic simplifications induced by the equations to obtain a new $S' = \{C'_1, \ldots, C'_k\}$. By "obvious" we do not yet mean reductions based on Gröbner bases, but instead have in mind trivial substitutions and simplifications like those we saw with the equational branches of split inequalities in ψ above.

- After the substitutions and simplifications, many of the conjunctions C'_i may be trivially recognisable as inconsistent. For example, if χ was our ψ treated above,

then some of the C'_i will have $(0 = 1)$ as a conjunct. We should perform some "simple reasoning," similar to that we did before splitting any inequalities, to filter out inconsistent conjunctions which are so easy to recognise. Let S'' be the resulting set of C'_i's which we simplified but did not easily refute.

- Then, for each C'_i in S'', we will do the following:
  - Let $Strict(C'_i)$ be the conjunctive formula containing all *strict* inequality atoms of C'_i. Let $\mathcal{E}(C'_i)$ be the collection of equations of C'_i.
  - If $Strict(C'_i) = C'_i$ so that $\mathcal{E} = \emptyset$, then we may simply run full-dimensional partial CAD on C'_i and if it terminates, then the result will be an (un)satisfiability decision which is correct for C'_i. In particular, if we conclude any C'_i of this form to be satisfiable, then χ itself is satisfiable and we are done.
  - If $Strict(C'_i) \neq C'_i$, then things become more involved. We would like to be able to decide C'_i by full-dimensional partial CAD, but the problem of course is that both $Strict(C'_i)$ and $\left(\bigwedge_{p \in \mathcal{E}} p = 0\right)$ may be satisfiable independently, while their combination, C'_i, is unsatisfiable.
  - So, we will try to "inject" as much of the equational fragment of C'_i as we can into the strict inequality fragment, and then attempt to refute C'_i by full-dimensional partial CAD.
  - To do this, we will use Gröbner bases as follows:
    - Choose a monomial order $\prec$,
    - Compute a Gröbner basis induced by $\mathcal{E}$ as $G = GB_{\prec}(\mathcal{E})$,
    - Reduce every polynomial in $Strict(C'_i)$ by G to obtain $Red_G(Strict(C'_i))$ (this is the "injection" of aspects of the equational fragment into the non-strict inequalities),
    - Finally, use full-dimensional partial CAD to decide $Red_G(Strict(C'_i))$. If the answer is `unsat`, then we may conclude that C'_i itself is unsatisfiable. If the answer is `sat`, then we can not in general trust this (of course, if the full-dimensional partial CAD procedure gives us a satisfying sample point for $Red_G(Strict(C'_i))$, then we can check to see if that sample point also satisfies C'_i, and if so we may conclude C'_i (and hence χ) to be satisfiable.)
- If at this point χ has not been decided, then we will run normal CAD on any C'_i whose (un)satisfiability has not been determined.

It is easy to build a strategy of this sort in **RAHD**. A simple initial approach is contained as the strategy named `calculemus-0` in **Figure 8.6.1.2**. This strategy, which we will soon refine, is referred to as `calculemus-0` and is roughly the same[2] strategy that we used in our paper [PJ09]. This strategy follows our approach outlined above:

1. The CMF `split-ineqs(max-splits := 12)` splits up to twelve non-strict inequalities and generates a subgoal consisting of some collection of cases. (Note that we will execute this strategy using "recursive subgoaling," i.e., the strategy will be called recursively upon any subgoals generated during its execution).

2. Many light-weight reasoning mechanisms are applied to simplify the formula and eliminate inconsistent cases / recognise satisfying ones if they exist.

3. The CMF `rcr-ineqs` rewrites polynomials appearing in inequalities in each case w.r.t. a Gröbner basis induced by the equations in the case. This "injects" some of the equational structure of each case into the (possibly strict) inequality fragment.

4. The CMF `qepcad(open := 1)` uses QEPCAD-B to perform full-dimensional partial CAD only upon the strict inequality fragment of each case. In general, only judgments of unsatisfiability are trusted here.

5. Finally, if any cases remain undecided, then normal partial CAD is performed upon them.

Table 8.6.1.2 shows the performance of the `calculemus-0` **RAHD** strategy (and two of its refinements, which we will come to shortly) on the twenty-four example problems[3] considered in [PJ09] (cf. **Appendix A**) and compares this performance to that of QEPCAD-B and two quantifier elimination procedures available in Reduce/Redlog:

- Rlqe, which is an enhanced implementation by Dolzmann and Sturm of Weispfenning's quadratic virtual term substitution (VTS) [Wei97] , and

[2] Though back then **RAHD** did not yet have its own strategy language. Instead, there was just a single heuristic proof strategy hard-coded in the system. This hard-coded procedure was the one reported upon in [PJ09].

[3] A reader who consults our earlier published paper [PJ09] may notice that the running times we present for `calculemus-0` in this chapter are significantly faster than those reported in the paper for what is essentially the same proof strategy. This is because since that publication, we have radically improved the integration of QEPCAD-B with **RAHD**, so that repeated applications of QEPCAD-B within the same strategy execution are now much more efficient than they used to be.

Table 8.1: The three **RAHD** `calculemus` proof strategies compared with QEPCAD-B and Redlog on twenty-four problems.

	dim	deg	div	calc-0	calc-1	calc-2	qepcad-b	redlog/rlqe	redlog/rlcad
P0	5	4	N	.91	1.59	1.7	416.45*	40.4	-
P1	6	4	N	1.69	3.08	3.42	-*	-	-
P2	5	4	N	1.34	2.41	2.62	-*	-	-
P3	5	4	N	1.52	2.56	2.75	-*	-	-
P4	5	4	N	1.14	2.02	2.16	-*	-	-
P5	14	2	N	.25	.26	.27	-*	97.4	-
P6	11	5	N	147.4	.07	.06	-*	<.01	<.01
P7	8	2	N	.05	<.01	<.01	.08	<.01	<.01
P8	7	32	N	4.5	.1	<.01	8.38	<.01	-
P9	7	16	N	4.51	.15	<.01	.29	.01	6.7
P10	7	12	N	100.74	20.76	8.85	-*	-	-
P11	6	2	Y	1.6	.5	.53	.01	.01	.05
P12	5	3	N	.78	.3	.36	.02	.01	.07
P13	4	10	N	3.83	3.95	4.02	-*	-	-
P14	2	2	N	4.55	1.67	.07	.01	-	-
P15	4	3	Y	.177	.2	.12	.01	<.01	<.01
P16	4	2	N	9.99	2.17	2.1	.02	<.01	<.01
P17	4	2	N	.62	.59	.65	.28	.02	.61
P18	4	2	N	1.25	1.28	1.27	.01	<.01	<.01
P19	3	6	Y	3.34	1.72	2.08	.02	.01	.7
P20	3	4	N	1.18	.65	.65	.01	<.01	.3
P21	3	2	N	.02	.03	<.01	.02	.01	.1
P22	2	4	N	<.01	<.01	<.01	.01	<.01	<.01
P23	2	2	Y	<.01	<.01	<.01	<.01	<.01	<.01

Explanation of columns:

High-level problem features: [**dim**] dimension, [**deg**] maximal total multivariate degree of polynomials, [**div**] whether or not problem contains division operator.

Timing: (in seconds)

A mark of (-) in any of the timing columns means the system listed was unable to solve the problem in 600 seconds. A mark of (*) in the **QB** column means that QEPCAD-B's default resource settings were raised in order to avoid reaching resource limits.

For problems involving division, the Redlog translation flag RLNZDEN was used both for Rlqe and Rlcad runs as well as for generating the multiplicative translations of the problems for QEPCAD-B.

- Rlcad, which is an implementation by Seidl, Dolzmann and Sturm of Collins-Hong's partial CAD [DSS04].

Experiments were performed on a 2 x 2.4 GHz Quad-Core Intel Xeon PowerMac with 10GB of 1066 MHz DDR3 RAM.

The full listing of the problems considered in **Table 8.6.1.2** — including translations of the problems into the input formats for each tool — may be obtained (cf. **Appendix A**).

For now, let us only compare `calculemus-0` with the QEPCAD-B and Redlog procedures. With this restriction, the results of these experiments can be broadly summarized as follows:

- The `calculemus-0` strategy is able to solve a number of high-dimension, high-degree problems that QEPCAD-B, Redlog/Rlqe, and Redlog/Rlcad are not. (To our knowledge, no other system besides **RAHD** has been able to solve problems P1, P2, P3, P4, P10 and P13). It is interesting that while the `calculemus-0` strategy involves an exponential blow-up in its reliance on inequality splitting followed by a DNF normalisation, for many problems the increase in complexity caused by this blow-up is overshadowed by the decrease in complexity of the CAD-related computations this process induces.

- Redlog/Rlqe is able to solve a number of high-dimension, high-degree problems that QEPCAD-B and Redlog/Rlcad are not.

- Redlog/Rlqe is able to solve a number of problems significantly faster than the `calculemus-0` strategy, Redlog/Rlcad, and QEPCAD-B.

- For the problems QEPCAD-B is able to solve directly, using QEPCAD-B directly tends to be much faster than using the `calculemus-0` strategy.

Indeed, these were the conclusions we reached at the end of our paper [PJ09]. There, we made the following observation and proposal for future work:

> Since QEPCAD-B outperforms [the **RAHD** `calculemus-0`] strategy on many low-dimension, low-degree problems, we should develop heuristics that use structural features of a problem to evaluate *a priori* its suitability for a direct handling by QEPCAD-B, causing [it] in those cases to bypass both its inequality splitting [..] and all other CMFs in the [strategy].

At the time of writing that statement, making such heuristic changes would have been a substantial undertaking. Now, with **RAHD**'s strategy language, experimenting with heuristic changes of this nature is easy.

Based upon interactive exporation of the twenty-four benchmark problems, we developed two simple extensions of the `calculemus-0` strategy. We show these in **Figure 8.6.1.2**. Their performance is compared in **Table 8.6.1.2**.

Overall, the final refinement, `calculemus-2`, substantially improves upon the strategy `calculemus-0` on problems P6, P8, P10, P11, P12, P14, P16, P19 and P20, often by many orders of magnitude. On problems P0, P1, P2, P3, P4, `calculemus-2` is slower than `calculemus-0` by roughly a factor of two. Strategies `calculemus-1` and `calculemus-2` are roughly equal for most problems, except for P1 and P19 where `calculemus-2` is slightly ($\cong$ 10-20%) slower, and P10 and P14 where `calculemus-2` is substantially ($\cong$ 2-25x) faster. Note that the five problems for which `calculemus-2` is significantly (1-2x) worse than `calculemus-0` are "hard" in the sense that they are not solved by all (indeed, usually by any) of QEPCAD-B, Redlog/Rlqe and Redlog/Rlcad.

This final strategy is able to solve a number of high-dimension, high-degree problems beyond the reaches of QEPCAD-B and the Redlog procedures, and is only consistently worse in timing than QEPCAD-B and Redlog on small problems which could already be solved by QEPCAD-B and Redlog almost instantly ($\leq$ .02 seconds).

In **Appendix B**, we give detailed profiling data on the relevant **RAHD** executions of these strategies. By examining this data, one can better understand the reasons underlying the performance differences between them. Let us examine one of the more interesting tables given there, the data for P10. We recapitulate its profiling table in **Figure 8.6.1.2** for convenience. Also, recall the qualitative content of the difference between the three strategies on P10 in **Figure 8.6.1.2**: Namely, `calculemus-2` performs significantly better than `calculemus-1` which performs significantly better than `calculemus-0`.

First, we see that the size of the proof tree is not constant among the three strategies. It contains 8,129 cases for `calculemus-1` and `calculemus-2`, but 32,768 for `calculemus-0`. This is because whenever `calculemus-0` is run upon a goal, it unconditionally splits up to 12 non-strict inequalities in each case in the goal. If this splitting succeeds, then this generates subgoals, and the strategy is recursively executed upon them, which may lead to more splitting. Contrast this with the latter two strategies: `calculemus-1` guards inequality splitting with the measure-value conditional `(gd =`

```
defstrat calculemus-0
  [split-ineqs(max-splits := 12); simp-zrhs; run stable-simp; demod-lin;
   run stable-simp; simp-real-null; fert-tsos; univ-sturm-ineqs;
   satur-lin; triv-ideals; run stable-simp; rcr-ineqs; run stable-simp;
   fert-tsos; run stable-simp; simp-zrhs; int-dom-zpb; rcr-ineqs;
   qepcad(open? := 1); qepcad].


defstrat calculemus-1
  [[when (gd = 0) [split-ineqs(max-splits := 12)]];
   interval-cp(max-contractions := 10); simp-zrhs; run stable-simp;
   demod-lin; run stable-simp; simp-real-null; fert-tsos; univ-sturm-ineqs;
   satur-lin; interval-cp; triv-ideals; run stable-simp; interval-cp;
   rcr-ineqs; run stable-simp; fert-tsos; run stable-simp; interval-cp;
   simp-zrhs; interval-cp; int-dom-zpb; rcr-ineqs;
   when (dim <= 7 /\ deg <= 30) [qepcad(open? := 1); qepcad]].

defstrat calculemus-2
  [interval-cp(max-contractions := 10);
   [when (dim <= 3 /\ deg <= 3) [qepcad]];
   [when (gd = 0 /\ dim >= 2) [split-ineqs(max-splits := 12)]];
   interval-cp(max-contractions := 20); simp-zrhs; run stable-simp;
   demod-lin; run stable-simp; simp-real-null; fert-tsos; univ-sturm-ineqs;
   satur-lin; interval-cp; triv-ideals; run stable-simp; interval-cp;
   rcr-ineqs; run stable-simp; fert-tsos; run stable-simp; interval-cp;
   simp-zrhs; interval-cp; int-dom-zpb; rcr-ineqs;
   when (dim <= 7 /\ deg <= 30) [qepcad(open? := 1); qepcad]].
```

Figure 8.14: Experimental strategies `calculemus-0`, `calculemus-1` and `calculemus-2`.

`0)` and `calculemus-2` by `(gd = 0 /\ dim >= 2)`. In both, the requirement that `(gd = 0)` (recall that `gd` is goal depth) precludes inequality splitting for cases in the goalset of any non-toplevel goal. In addition, `calculemus-2` first, before any inequality splitting takes place, tries to eliminate cases using both interval constraint propagation (unguarded but with a low contraction bound) and guarded partial CAD.

P10	calc-0	calc-1	calc-2
#(Proof-tree)	32768	8192	8192
TRIV-IDEALS	278 (45.847)	96 (0.936)	21 (0.101)
SATUR-LIN	485 (1.465)	344 (0.756)	25 (0.275)
FERT-TSOS	70 (0.355)	48 (0.200)	8 (0.041)
DEMOD-LIN	1101 (0.238)	444 (0.048)	33 (0.003)
SIMP-GLS	34351 (1.290)	3384 (0.185)	33 (0.000)
SIMP-ZRHS	32919 (0.370)	3133 (0.031)	33 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	8194 (0.468)	2 (0.109)	2 (0.131)
QEPCAD (OPEN?:=1)	207 (21.081)	70 (10.881)	-
RCR-INEQS	180 (7.601)	70 (0.225)	-
INT-DOM-ZPB	0 (0.012)	0 (0.006)	-
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	5120 (6.052)	5120 (6.092)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	3039 (2.126)
DEMOD-NUM	32766 (3.318)	3070 (0.170)	33 (0.003)
SIMP-ARITH	1954 (0.430)	482 (0.143)	33 (0.006)
SIMP-REAL-NULL	142 (0.031)	0 (0.032)	0 (0.002)
UNIV-STURM-INEQS	4 (0.035)	0 (0.016)	0 (0.001)
INTERVAL-CP	-	178 (0.834)	4 (0.024)
	100.850	21.049	9.343

Figure 8.15: Fine-grained comparison of the calculemus strategies on P10. Please see **Appendix B** for how to read this table. The corresponding tables for the rest of the problems may also be found in that appendix.

Next, let us look at where each strategy spends most of its time. (See **Section 8.5** for more detail on all of the CMFs referenced during this discussion.) For `calculemus-0`, the most time is spent in the CMFs `TRIV-IDEALS`, `QEPCAD(OPEN?:=1)`, and `RCR-INEQS`. Recall that in these strategies, `RCR-INEQS` is used to rewrite the polynomials appearing in inequalities in a case w.r.t. a Gröbner basis induced by the case equations. The strict inequality fragments of the cases resulting from this rewriting are eventually examined using a full-dimensional variant of partial CAD, `QEPCAD(OPEN?:=1)`. By doing such a large amount of inequality splitting, and doing so without help from techniques such as interval constraint propagation which might quickly eliminate a huge collection of the generated cases, `calculemus-0` is forced to apply this combination of `RCR-INEQS` and `QEPCAD(OPEN?:=1)` to 207 cases, which is many more than required by the other strategies. Both of these, especially partial CAD, are expensive, as is `TRIV-IDEALS` which also requires Gröbner basis construction. Let us now look at `calculemus-1` and `calculemus-2`.

First, their guarded splitting causes them to have much fewer cases in their proof trees. For `calculemus-1`, the most CMF time is spent with `QEPCAD(OPEN?:=1)` and `INTERVAL-CP(MAX-CONTRACTIONS:=10)`. Crucially, by using two forms of interval constraint propagation preceded by saturation with linear orientations (`SATUR-LIN`), `calculemus-1` is able to eliminate roughly 65% of its open cases in less than 7 seconds. The remaining cases are then solved by a number of methods. The vast majority of them are eliminated through arithmetical simplification, the recognition of trivial sums of squares, and the recognition of trivial ideals. This takes in total less than 2 seconds. Finally, the remaining 70 cases are eliminated using a combination of `RCR-INEQS` and `QEPCAD(OPEN?:=1)`. Note that 70 is much smaller than 207, the number of cases which were eliminated in this way by `calculemus-0`. Indeed, `calculemus-0` spends roughly 29 seconds on this combination while `calculemus-1` spends roughly 11 seconds.

Finally, let us look to `calculemus-2`. This strategy generates the same number of cases as `calculemus-1`, but processes them differently. First, we see that the most time in `calculemus-2` is spent doing interval constraint propagation. Crucially, `calculemus-2` applies an additional instance of interval methods beyond that done by `calculemus-1`. Both strategies use `INTERVAL-CP(MAX-CONTRACTIONS:=10)` to eliminate 5,120 of their 8,192 cases. But, the additional interval method used by `calculemus-2`, `INTERVAL-CP(MAX-CONTRACTIONS:=20)`, is able to eliminate 3,039 more cases in roughly 2 seconds. These cases had to be eliminated by `calculemus-1`

using other, and often more expensive, methods. The strategy `calculemus-2` is then able to make use of a battery of very cheap CMFs to eliminate its remaining cases, without every needing to apply `RCR-INEQS` or `QEPCAD(OPEN?:=1)`.

8.6.1.3 Experimental Conclusion

Let us conclude this class of experiments with a few observations. First, after performing for the remaining problems an analysis similar to that we did for P10 above (cf. **Appendix B** for the data), we see that there are many trade-offs when constructing strategies. On the one hand, by enhancing the complexity of a strategy so that it performs a more intricate application of CMFs with more nuanced and carefully tuned guards, one can at times much improve the execution of the strategy, especially on targeted classes of hard problems. On the other hand, increasing the complexity of a strategy can make the entire strategy application process more expensive. As we have seen with many problems such as P10 that we analysed above, this increase in complexity of a strategy can have major payoffs. But, as we have also seen with problems such as P1, this increase in complexity can make more nuanced variations of strategies significantly slower than their simpler counterparts. Because of this, it seems very important to have a tool like **RAHD** which gives the researcher an ability to easily experiment with strategy variations.

More broadly, through this class of experiments we have given evidence that a heterogenous collection of $\exists$ **RCF** proof procedures can be carefully combined in such a way as to make previously out of reach problems soluble by automatic methods. In particular, we have shown how proof methods based on Gröbner bases and full-dimensional partial CAD may be compellingly combined. We have illustrated how one does this using the methodology **RAHD** was designed to facilitate: Namely, one begins with a difficult problem or class of similar of problems, uses **RAHD** interactively to explore the problem structure and find a custom proof method which succeeds, and then uses **RAHD**'s proof strategy language to construct an automatic proof procedure utilising this strategy.

8.6.2 A Concrete Abstract Partial CAD Instantiation

For this experiment, we will build a concrete instance of our Abstract Partial CAD framework and study its use in detail upon an example $\exists$ **RCF** formula. We will contrast the processing of this AP-CAD decision algorithm with that of classical CAD and

partial CAD algorithms, observing how they differ during the lifting or stack construction phase. Finally, we will present the results of a similar comparison upon a number of additional $\exists$ **RCF** formulas. Please see **Chapter 7** for an account of AP-CAD, its motivations, definitions, and the axioms an AP-CAD stage must obey.

To build an instance of AP-CAD, we need to construct an AP-CAD *n-theatre*. Such a theatre will be a function from $\mathbb{N}$ to the collection of AP-CAD *stages*.

Let $i \in \mathbb{N}^+$ be arbitrary and let us abbreviate the set of all finite sets of i-dimensional real vectors (i.e., the set of all possible sets of i-dimensional sample points) as

$$R_i = \{s \subset \mathbb{R}^i \mid |s| < \omega\}.$$

Recall that an AP-CAD stage is a triple

$$\langle \langle \mathbb{S}, w \rangle, \mathbb{F}, \mathbb{P} \rangle$$

where

$\langle \mathbb{S}, w \rangle$ is a *cell selection strategy* consisting of

a *cell selection function* $\mathbb{S} : R_i \times \mathbb{N} \to R_i$,

a *covering width function* $w : R_i \to \mathbb{N}$,

$\mathbb{F} : \mathcal{L}_{\exists OR} \times R_i \to \mathcal{L}_{\exists OR}$ is a *formula construction function*, and

$\mathbb{P} : \mathcal{L}_{\exists OR} \to \{\textbf{true}, \textbf{false}, \textbf{unknown}\} \cup \bigcup_{j \in \mathbb{N}^+} \mathbb{R}^j$ is an $\exists$ **RCF** *proof procedure*.

We will build an AP-CAD theatre based upon a "divide and conquer" strategy for applying interval arithmetic to eliminate sample points during the lifting phase of partial CAD construction. We will use the **RAHD** proof strategy language to construct the $\exists$ **RCF** proof procedure $\mathbb{P}$.

8.6.2.1 Defining the AP-CAD Theatre

In defining this theatre, it will be useful to allow our functions to work explicitly over *lists* of sample points as opposed to sets of sample points. To do so, we use the maps

$$StoL : R_i \to Lists(R_i)$$

and

$$LtoS : Lists(R_i) \to R_i.$$

StoL will map a set of sample points to a *sorted* representation of the set as a list, and *LtoS* will map a list of sample points to its underlying set. We use the lexicographic product order of the normal ordering $<$ on $\mathbb{R}$ to order the sample points. If l is a list, then $|l|$ will be the length of the list. If l is a list and $0 \leq m \leq n \leq |l|$, then $subseq(l,m,n)$ will be the subsequence of l of the form[4] $\langle l(m), \ldots, l(n-1)\rangle$.

We build now a stage for our theatre.

cell selection function $\mathbb{S}(s,n) = LtoS(\mathbb{S}_{Lists}(StoL(s),n))$ where

$$\mathbb{S}_{Lists}(l,n) = \begin{cases} l & \text{if } n \leq 1, \\ \lfloor S_{Lists}(l,k) \rfloor & \text{if } n = 2k, \\ \lceil S_{Lists}(l,k) \rceil & \text{if } n = 2k+1, \end{cases}$$

and

$$\lfloor l \rfloor = \begin{cases} subseq(l,0,k) & \text{if } |l| = 2k, \\ subseq(l,0,k+1) & \text{if } |l| = 2k+1, \end{cases}$$

and

$$\lceil l \rceil = \begin{cases} subseq(l,k,|l|) & \text{if } |l| = 2k, \\ subseq(l,k+1,|l|) & \text{if } |l| = 2k+1. \end{cases}$$

Let us explain these functions in words. The function $\lfloor l \rfloor$ returns the first half of the list l if $|l|$ is even, and returns the first $k+1$ elements of l if $|l| = 2k+1$. The function $\lceil l \rceil$ returns the second half of the list l if $|l|$ is even, and returns the final k elements of l if $|l| = 2k+1$. In this way, we always have that the concatenation of $\lfloor l \rfloor$ and $\lceil l \rceil$ is l itself. These two functions are used to "bisect" the list l by the function $\mathbb{S}_{Lists}$, regardless of whether or not $|l|$ is even or odd.

The function $\mathbb{S}_{Lists}(l,n)$ computes subsequences of the list l in a "divide and conquer" fashion, with the parameter n specifying which subsequence should be computed. It is best understood as representing an enumeration of subsequences of l which have been situated in a binary tree. To illustrate a concrete example, let $l = \langle a_1, \ldots, a_7\rangle$. Then, we have

$$\mathbb{S}_{Lists}(l,1) = l = \langle a_1, \ldots, a_7\rangle,$$
$$\mathbb{S}_{Lists}(l,2) = \lfloor \langle a_1, \ldots, a_7\rangle \rfloor = \langle a_1, \ldots, a_4\rangle,$$

[4] This perhaps strange way of indexing list subsequences is used so that our description matches our actual implemention, as this is how Common Lisp does list subsequencing via the `subseq` function. For example, `subseq(`$\langle a,b,c\rangle$`, 0, 2)` $= \langle a,b\rangle$ and `subseq(`$\langle a,b,c\rangle$`, 0, 0) = nil`, the empty list.

$$\mathbb{S}_{Lists}(l,3) = \lceil \langle a_1,\ldots,a_7\rangle \rceil = \langle a_5,\ldots,a_7\rangle,$$
$$\mathbb{S}_{Lists}(l,4) = \lfloor \lfloor \langle a_1,\ldots,a_7\rangle \rfloor \rfloor = \lfloor \langle a_1,\ldots,a_4\rangle \rfloor = \langle a_1,a_2\rangle,$$
$$\mathbb{S}_{Lists}(l,5) = \lceil \lfloor \langle a_1,\ldots,a_7\rangle \rfloor \rceil = \lceil \langle a_1,\ldots,a_4\rangle \rceil = \langle a_3,a_4\rangle,$$
$$\mathbb{S}_{Lists}(l,6) = \lfloor \lceil \langle a_1,\ldots,a_7\rangle \rceil \rfloor = \lfloor \langle a_5,\ldots,a_7\rangle \rfloor = \langle a_5,a_6\rangle,$$
$$\mathbb{S}_{Lists}(l,7) = \lceil \lceil \langle a_1,\ldots,a_7\rangle \rceil \rceil = \lceil \langle a_5,\ldots,a_7\rangle \rceil = \langle a_7\rangle.$$

The cell selection function $\mathbb{S}(s,n)$ then maps s to an underlying sorted list representation $StoL(s)$ and uses $\mathbb{S}_{Lists}$ to compute the nth subsequence of $StoL(s)$ with respect to the "divide and conquer" enumeration order given above.

covering width function We will use a constant covering width function w of the form

$$w(s) = 3.$$

Given a collection of sample points s, this covering width will cause the APCAD lifting algorithm (cf. **Algorithm 7.6.3**) to attempt to eliminate the cell selections $\mathbb{S}(s,1)$ through $\mathbb{S}(s,3)$. (This is quite a "shallow" depth for a "divide and conquer" strategy. Nevertheless, it will be useful for keeping our explicit examples small enough to discuss in detail and can be easily changed if one wishes to experiment with variations of it.)

formula construction function Our formula construction function $\mathbb{F} : \mathcal{L}_{\exists OR} \times R_i \to \mathcal{L}_{\exists OR}$ will accept an $\exists$ **RCF** formula φ and a set of i-dimensional sample points s and work as follows:

1. Let $min_j(s)$ be the minimal value ever appearing as coordinate j in a sample point in s. To be precise,

$$min_j(s) = min\{\pi_j(x) \mid x \in s\},$$

where π_j projects a sample point $x \in \mathbb{R}^i$ onto its jth coordinate.

2. Similarly, let $max_j(s)$ be s.t.

$$max_j(s) = max\{\pi_j(x) \mid x \in s\}.$$

3. Then,

$$\mathbb{F}(\varphi,s) = \exists \vec{x} \left[\left(\bigwedge_{j=1}^{i} x_j \geq min_j(s) \ \wedge \ x_j \leq max_j(s) \right) \ \wedge \ QF(\varphi) \right].$$

∃ RCF proof procedure

We use an ∃ **RCF** procedure — expressed as a **RAHD** proof strategy — which performs simple formula simplification, saturation of linear bounds on variables, followed by interval constraint propagation (cf. **Section 8.5** for more information on CMFs):

```
[ simp-zrhs; run stable-simp; satur-lin;
  interval-cp(max-contractions := 30)].
```

RAHD's execution of this proof strategy then gives rise to our AP-CAD stage's ∃ **RCF** proof procedure $\mathbb{P}$.

Lemma 8.6.1. *$\langle\langle \mathbb{S}, w\rangle, \mathbb{F}, \mathbb{P}\rangle$ as defined above is an AP-CAD stage.*

Proof. As **RAHD** guarantees that the execution of its proof strategies correspond to proper AP-CAD ∃ **RCF** proof procedures, the only non-trivial property to verify is that our formula construction function $\mathbb{F}$ satisfies the *relevance judgment* axioms. Let φ be an $\mathcal{L}_{\exists OR}$ formula in $x_1, \ldots, x_n$ and $s \subset \mathbb{R}^i$ a finite set of i-dimensional sample points $(1 \leq i \leq n)$.

We must verify that

$$\mathbf{RCF} \models \neg\mathbb{F}(\varphi, s) \quad \Longrightarrow \quad \mathcal{N}(\varphi, s),$$

and

$$\mathbf{RCF} \models \mathbb{F}(\varphi, s) \quad \Longrightarrow \quad \mathbf{RCF} \models \varphi,$$

where (restating the property a bit more concretely than its original axiomatisation in **Section 7.6.1**):

1. $\mathcal{N}(\varphi, s)$ means that no child (at any ancestral depth, i.e., in a P_{i+1}-invariant CAD of $\mathbb{R}^{i+1}$, in a P_{i+2}-invariant CAD of $\mathbb{R}^{i+2}, \ldots$, in a P_n-invariant CAD of $\mathbb{R}^n$) of any sample point in s will satisfy $QF(\varphi)$.

In the first case, we have that any child of any sample point in s will satisfy

$$\left(\bigwedge_{j=1}^{i} x_j \geq min_j(s) \ \wedge \ x_j \leq max_j(s) \right),$$

and so

$$\mathbf{RCF} \models \neg\mathbb{F}(\varphi, s) \quad \Longrightarrow \quad \mathcal{N}(\varphi, s)$$

obviously holds. In the second case,

$$\mathbf{RCF} \models \mathbb{F}(\varphi, s) \quad \Longrightarrow \quad \mathbf{RCF} \models \varphi$$

is immediate. □

Finally, again to keep our detailed examples below from becoming too large, we will turn this AP-CAD stage $\langle\langle \mathbb{S}, w\rangle, \mathbb{F}, \mathbb{P}\rangle$ into an AP-CAD theatre $\mathbb{T}$ in a trivial fashion:

$$\mathbb{T}(n) = \langle\langle \mathbb{S}, w\rangle, \mathbb{F}, \mathbb{P}\rangle.$$

That is, the same stage $\langle\langle \mathbb{S}, w\rangle, \mathbb{F}, \mathbb{P}\rangle$ will be used at every dimension during AP-CAD lifting.

8.6.2.2 Applying AP-CAD in Detail

Let us now apply our concrete AP-CAD theatre to some example $\mathcal{L}_{\exists OR}$ formulas and examine its execution. Recall that the decision method proceeds in four steps: projection, base, lifting and evaluation. We examine each step in detail.

Let φ be as follows:

$$\varphi = \left[\begin{array}{r} \exists x_1\ \exists x_2\ \exists x_3\ \exists x_4 \\ (x_1x_4 + x_2x_4 + x_3x_2 < 0) \\ \wedge\ (x_2 > 0)\ \wedge\ (x_3 > 0)\ \wedge\ (x_4 > 0) \\ \wedge\ (x_3x_4 - x_4^2 + x_3^2 + 1 < 0) \end{array}\right].$$

As φ is an $\exists$ **RCF** sentence s.t. $QF(\varphi)$ consists of a conjunction of strict polynomial inequalities, it follows by McCallum's Theorem (cf. **Section** 7.6.2) that we may decide φ by only examining full-dimensional cells during partial CAD construction. This allows us avoid irrational algebraic number computations, as full-dimensional cells (i.e., *sectors*) always contain rational points. This also permits us to use the Brown-McCallum projection operator (cf. **Definition** 7.6.5) to obtain our CAD projection factor sets, which can lead to much smaller projection sets than those obtained with projection operators valid for general $\exists$ **RCF** formulas.

We will now walk through using AP-CAD to decide φ. It will turn out that φ is in fact **true** over **RCF**, and we will illustrate the process of constructing a witness to its

truth. First, we will compute the projection (factor) sets for φ. Then, we will compute the base phase for our level 1 projection set. Finally, we will show how four different variants of CAD operate during the *lifting* phase. These variants are:

1. full-dimensional lifting without eliminating any full-dimensional cells,
2. full-dimensional lifting with standard partial CAD used to eliminate cells,
3. full-dimensional lifting with our AP-CAD stage used to eliminate cells,
4. full-dimensional lifting with a combination of standard partial CAD and our AP-CAD stage used to eliminate cells.

With each progressive variant, the number of cells eliminated during lifting will change. The final variant will be the best in the sense that it will allow us to eliminate the most cells during partial CAD construction.

8.6.2.2.1 Projection Sets We first extract the polynomials of φ, which gives us $P_4 \subset \mathbb{Z}[x_1,x_2,x_3,x_4]$:

$$P_4 = \left\{ \begin{array}{r} x_2, x_3, x_4, x_1x_4 + x_4x_2 + x_3x_2, \\ x_4^2 - x_3x_4 - x_3^2 - 1 \end{array} \right\}.$$

Then, we apply the Brown-McCallum projection operator $BMProj : \mathbb{Z}[x_1,\ldots,x_{i+1}] \rightarrow \mathbb{Z}[x_1,\ldots,x_i]$ to obtain $P_3 \subset \mathbb{Z}[x_1,x_2,x_3]$:

$$BMProj(P_4) = P_3 = \left\{ \begin{array}{r} x_2, x_3, -x_1^2x_3^2 - x_1x_3^2x_2 + x_3^2x_2^2 - x_1^2 - 2x_1x_2 - x_2^2, \\ x_3^2 + 1, x_1 + x_2, 5x_3^2 + 4 \end{array} \right\}.$$

We apply $BMProj$ again to obtain $P_2 \subset \mathbb{Z}[x_1,x_2]$:

$$BMProj(P_3) = P_2 = \left\{ -x_1^2 - x_1x_2 + x_2^2, x_1 + 3x_2, x_1 + 2x_2, x_1 + x_2, x_2 \right\}.$$

Finally, we apply $BMProj$ one last time to obtain $P_1 \subset \mathbb{Z}[x_1]$:

$$BMProj(P_2) = P_1 = \left\{ x_1 \right\}.$$

It is worth stating that obtaining a level-1 projection factor set P_1 s.t. $|P_1| = 1$ is quite unusual. Even for this small problem, the size of the respective projection sets can change drastically depending upon the projection variable order used. Nevertheless, the example arising through the use of this projection order is nice as it results in constructions small enough so that a detailed description of the decision process can be given quite compactly.

8.6.2.2.2 Base Phase We now compute the base collection of sample points of $\mathbb{R}^1$ induced by our level 1 projection set $P_1 \subset \mathbb{Z}[x_1]$. As our P_1 is rather uncharacteristically a singleton, this is trivial in this particular example. But, let us state what one must do in general for the full-dimensional base phase, so that our walk-through is applicable when P_1 is larger. We will then follow this same sample-point computation process when constructing stacks over cells in the lifting phase. We will build our collection of sample points in the following manner:

1. We process P_1 into a new set $\mathit{CoPrime}(P_1) \subset \mathbb{Z}[x_1]$ so that no two distinct polynomials $p, q \in \mathit{CoPrime}(P_1)$ share a root, while maintaining the invariant that

$$\{r \in \mathbb{R} \mid \exists p \in P_1 (p(r) = 0)\} = \{r \in \mathbb{R} \mid \exists p \in \mathit{CoPrime}(P_1)(p(r) = 0)\}.$$

   This can be done using univariate GCD and division. Our P_1 in this example happens to already have this property, so we simply set $\mathit{CoPrime}(P_1) = P_1$.

2. We apply univariate real root isolation to the polynomials in $\mathit{CoPrime}(P_1)$ to obtain a collection of pairwise disjoint compact real intervals $I_1, \ldots, I_k \subset \mathbb{R}$ s.t. every real root of a polynomial $p \in \mathit{CoPrime}(P_1)$ is contained in exactly one interval I_i, and for each interval I_i, there exists only one $p \in \mathit{CoPrime}(P_1)$ s.t. I_i contains a real root of p (here we exploit the fact that no distinct $p, q \in \mathit{CoPrime}(P_1)$ share a root). This gives us a bijection

$$i \,:\, \{r \in \mathbb{R} \mid \exists p \in P_1 (p(r) = 0)\} \;\rightarrow\; \{I_1, \ldots, I_k\}$$

   s.t.

$$\forall r \in \mathbb{R} \, (\exists p \in P_1 (p(r) = 0) \implies r \in i(r)) \, .$$

   This $\{I_1, \ldots, I_k\}$ is called an *isolating set of intervals* for the roots of P_1.

3. Because of their pairwise disjointness, $I_1, \ldots, I_k$ have a natural ordering determined, for instance, by comparing their lower-bound components. WLOG, assume

$$I_1 < I_2 < \ldots < I_k.$$

   This gives a "sketch" of a CAD of $\mathbb{R}^1$, with each interval I_i giving an approximation to a 0-dimensional cell (a *section*) which consists only of a root of a polynomial in P_1. If we were performing normal CAD without exploiting McCallum's Theorem, we would have to exactly represent these 0-dimensional cells, which may be irrational algebraic numbers, and construct stacks over them. The

1-dimensional cells of the P_1-induced CAD of $\mathbb{R}^1$ are those in between each adjacent pair of 0-dimensional cells, before the 0-dimensional cell contained in I_1 and after the 0-dimensional cell contained in I_k. As our isolating intervals are pairwise disjoint, they give us enough information, without any further refinement, to select sample points in the 1-dimensional cells. Thankfully, this means that we do not have to exactly represent any of the 0-dimensional cells; our approximations of them given by $I_1, \ldots, I_k$ are good enough.

4. As we are only interested in full-dimensional cells, we only need sample points *in between* adjacent I_i's, before I_1 and after I_k. Since every such region we will be sampling is an open subset of $\mathbb{R}^1$, we can choose these sample points all to be rational points.

5. In our example, we choose the following sample points to form S_1, our base collection of sample points of $\mathbb{R}^1$ with one point taken from each full-dimensional cell of a P_1-invariant CAD of $\mathbb{R}^1$:

$$S_1 = \left\{ -1, 1 \right\}.$$

8.6.2.2.3 Four Variants of Lifting With the projection and base phases completed, we turn to the lifting phase of (partial) CAD construction. To illustrate the use of our concrete AP-CAD stage, we will show how the following four distinct approaches to lifting differ when applied to deciding φ:

1. full-dimensional lifting without eliminating any full-dimensional cells,

2. full-dimensional lifting with standard partial CAD used to eliminate cells,

3. full-dimensional lifting with our AP-CAD theatre used to eliminate cells,

4. full-dimensional lifting with a combination of standard partial CAD and our AP-CAD theatre used to eliminate cells.

It will turn out that as we consider them in sequence, each subsequent lifting method will exhibit quite different behaviour in terms of the number of cells eliminated. In the end, full-dimensional lifting with a combination of standard partial CAD and our AP-CAD theatre will allow us to decide φ in the most efficient manner.

8.6.2.2.4 Lifting Variant I: All Full-dimensional Cells In this first variant of lifting, we will construct the entire full-dimensional CAD. In general, it is structured as a tree of sample points, each drawn from a full-dimensional cell. But, since we are deciding a purely $\exists$ **RCF** formula, we can ignore the tree structure and arrange our representation as a collection of sets of sample points, with one set of sample points for each CAD level. (There is still an implicit tree structure, however, as a sample point $\langle r_1, r_2, r_3 \rangle \in \mathbb{R}^3$ will be seen as a "child" of the sample point $\langle r_1, r_2 \rangle \in \mathbb{R}^2$, for instance.) Since our φ is 4-dimensional, and as we are only sampling rational points, we will end up with four sets of sample points, $S_1 \subset \mathbb{Q}^1$ (which we have already computed), $S_2 \subset \mathbb{Q}^2$, $S_3 \subset \mathbb{Q}^3$ and $S_4 \subset \mathbb{Q}^4$. At times we will only describe salient features of these additional sets of sample points, instead of presenting them explicitly, as they become large. We construct them as follows:

($\mathbb{R}^1 \mapsto \mathbb{R}^2$): To lift from $\mathbb{R}^1$ to $\mathbb{R}^2$, we iterate over our base set of sample points S_1 as follows (recall $|S_1| = 2$):

For each $q \in S_1$,

1. Form the univariate family $P_2[x_1 \mapsto q]$ by substituting[5] q for x_1 in P_2,
2. Compute a "sketch" of a CAD of $\mathbb{R}^1$ induced by $P_2[x_1 \mapsto q]$ (in the same manner we construced a "sketch" of a CAD of $\mathbb{R}^1$ induced by P_1 above through univariate real root isolation and isolating intervals), and select rational sample points x_i from each of its full-dimensional cells. For each sample point $x_i \in \mathbb{Q}$ selected, we then form a 2-dimensional sample point through extending q by x_i, obtaining $\langle q, x_i \rangle \in \mathbb{Q}^2$. Let $S_{2,q}$ be the set consisting of these sample points of the form $\langle q, x_i \rangle$. ($S_{2,q}$ then represents a full-dimensional stack over q.)

Finally, our set of 2-dimensional sample points S_2 is the union of these $S_{2,q}$ as follows:

$$S_2 = \bigcup_{q \in S_1} S_{2,q}.$$

Given P_2 and S_1 as computed above during the projection and base phases of deciding φ, $S_2 \subset \mathbb{Q}^2$ computed in this way will be s.t.

$$|S_2| = 14.$$

[5]Note that as q may in general be in $(\mathbb{Q} \setminus \mathbb{Z})$, we may have that $P_2[x_1 \mapsto q] \subset (\mathbb{Q}[x_2] \setminus \mathbb{Z}[x_2])$. This turns out to not cause any problems, and indeed can be avoided altogether without changing the real affine variety induced by $P_2[x_1 \mapsto q]$ by multiplying through the resulting univariate polynomials by the denominators of their rational coefficients.

($\mathbb{R}^2 \mapsto \mathbb{R}^3$): We perform the next-dimensional analogue of the above procedure, this time working over the 14 sample points (each 2-dimensional) in S_2 and substituting them into P_3. After performing the relevant root isolation and sampling computations, this yields $S_3 \subset \mathbb{Q}^3$ s.t.

$$|S_3| = 40.$$

($\mathbb{R}^3 \mapsto \mathbb{R}^4$): Finally, we perform the next-dimensional analogue of the previous liftings, this time working over the 40 sample points (each 3-dimensional) in S_3 and substituting them into P_4. After performing the relevant root isolation and sampling computations, this yields $S_4 \subset \mathbb{Q}^4$ s.t.

$$|S_4| = 200.$$

So, lifting over every full-dimensional cell in our φ example ultimately results in having to compute 200 sample points in $\mathbb{Q}^4$, which will then be each substituted into $QF(\varphi)$ during the evaluation phase.

The coordinates of our sample points tend to become[6] more computationally unwieldy as we rise in dimension. For instance, here is one of these 200 sample points we computed for S_4.

$$\langle -1, -43/16, -119327/36200, 23133930249499/9896442880000 \rangle.$$

Here is one witness to φ contained in S_4, thus proving φ to be **true** over **RCF**:

$$\langle -1, 7/8, 501/410, 3917/410 \rangle.$$

Let us see how incorporating the methods of *partial* CAD during lifting can improve the situation by reducing the number of cells we must lift over.

8.6.2.2.5 Lifting Variant II: Classical Partiality In this variant of lifting, we will proceed in the manner of classical partial CAD (restricted to full-dimensional cells). This follows the basic algorithm described in **Section 7.6**. We recall the idea.

Beginning with our sample points $S_1 \subset \mathbb{Q}^1$ computed in the base phase, the "partiality" of this variant of lifting comes from the following process, which we follow for each $q \in S_1$: Before lifting over q, we will substitute q as a value for x_1 into $QF(\varphi)$ and

[6] We have some ideas for methods enabling us to select sample points with smaller bit-width than those we often select from sectors now. Pursuing this remains as future work beyond this thesis.

examine the truth of the resulting formula $QF(\varphi)[x_1 \mapsto q]$. If $QF(\varphi)[x_1 \mapsto q]$ can by ground evaluation and propositional reasoning (cf. **Section 7.6**) be seen to be unsatisfiable, then we will eliminate q and avoid lifting over it. Dually, if $QF(\varphi)[x_1 \mapsto q]$ can be seen to be satisfiable by polynomial arithmetic and propositional reasoning, then we can stop the CAD process altogether and judge φ to be **true**. If we happen to eliminate *all* of our sample points, then we can judge φ to be **false**. This "partiality" is then performed in the analogous manner when lifting to each successive dimension.

As discussed in **Section 7.6**, classical partial CAD requires a sample point selection strategy. When partially lifting from $\mathbb{R}^i$ to $\mathbb{R}^{i+1}$, this specifies an enumeration of S_i, the sample points for the relevant cells of $\mathbb{R}^i$. We use a simple cell selection mechanism below, given by ordering the members of S_i by the lexicographic extension of the normal $<$ relation of $\mathbb{R}$ and then selecting the sample points from left to right.

($\mathbb{R}^1 \mapsto \mathbb{R}^2$): Performing the partial CAD method as described above results in no elimination of members of S_1.

We then substitute these two points as values for x_1 in P_2, and perform the root isolation and full-dimensional sample point selection computations. As before, this results in a total of 14 sample points in $\mathbb{Q}^2$. We use the following points:

$$\left\{ \begin{array}{r} \langle 1,-2\rangle, \langle 1,-7/8\rangle, \langle 1,-17/32\rangle, \langle 1,-5/12\rangle, \langle 1,-1/6\rangle, \\ \langle 1,3/4\rangle, \langle 1,43/16\rangle, \langle -1,-43/16\rangle, \langle -1,-3/4\rangle, \langle -1,1/6\rangle, \\ \langle -1,5/12\rangle, \langle -1,17/32\rangle, \langle -1,7/8\rangle, \langle -1,2\rangle \end{array} \right\}.$$

($\mathbb{R}^2 \mapsto \mathbb{R}^3$): We perform the analogous partial lifting method for $\mathbb{R}^2 \mapsto \mathbb{R}^3$, this time working over our 14 sample points in $\mathbb{Q}^2$ computed above. In doing so, 7 out of the 14 sample points are eliminated. We are then left with only having to lift over the following set of 7 rational points in $\mathbb{Q}^2$:

$$\{\langle 1,3/4\rangle, \langle 1,43/16\rangle, \langle -1,1/6\rangle, \langle -1,5/12\rangle, \langle -1,17/32\rangle, \langle -1,7/8\rangle, \langle -1,2\rangle\}.$$

We then substitute these points, with each one giving values for x_1 and x_2, into P_3, and perform the root isolation and full-dimensional sample point selection computations. This results in a total of only 20 sample points in $\mathbb{Q}^3$.

($\mathbb{R}^3 \mapsto \mathbb{R}^4$): We perform the analogous partial lifting method for $\mathbb{R}^3 \mapsto \mathbb{R}^4$, this time working over our 20 sample points in $\mathbb{Q}^3$ computed above. In doing so, 10 out

of the 20 sample points are eliminated. We are then left with only having to lift over 10 rational points in $\mathbb{Q}^3$.

We then substitute these points, with each one giving values for x_1, x_2 and x_3, into P_4, and perform the root isolation and full-dimensional sample point selection computations. This results in a total of only 50 sample points in $\mathbb{Q}^4$.

Clearly, tremendous gains were made by employing partiality during lifting. Let us illustrate the differences between these first two lifting methods by comparing the cardinalities of the collections of sample points they retained at each dimension:

	Normal	Partial
$\mathbb{Q}^1$	2	2
$\mathbb{Q}^2$	14	7
$\mathbb{Q}^3$	40	10
$\mathbb{Q}^4$	200	50

8.6.2.2.6 Lifting Variant III: AP-CAD with Interval Theatre We now consider a lifting method which utilises our interval-based AP-CAD theatre defined previously. This will follow the "AP-CAD with Theatrical Lifting" algorithm (**Algorithm 7.6.3**) introduced in **Chapter 7**, but instantiated upon our concrete theatre $\mathbb{T}$.

Recall that we defined $\mathbb{T}$ to be s.t.

$$\forall n \in \mathbb{N}^+ \ (\mathbb{T}(n) = \langle\langle \mathbb{S}, w\rangle, \mathbb{F}, \mathbb{P}\rangle).$$

Thus, when performing AP-CAD lifting with $\mathbb{T}$, the same AP-CAD stage, $\langle\langle \mathbb{S}, w\rangle, \mathbb{F}, \mathbb{P}\rangle$ which we built to use some simple interval-based methods, will be applied at every dimension. The algorithm proceeds as follows:

$\mathbb{R}^1 \mapsto \mathbb{R}^2$: The covering width function w is applied to S_1 to yield $w(S_1) = 3$. This gives an upper-bound on the number of cell selections we will compute. As with the description of **Algorithm 7.6.3**, we use j to represent the "step" in the cell selection processing. Initially, j is set to 1. Next, the cell selection function $\mathbb{S}$ is applied to S_1 with a step value of 1, yielding:

$$\mathbb{S}(S_1, 1) = S_1 = \{-1, 1\}.$$

So, the entire set of base sample points has been selected.

Next, the formula construction function $\mathbb{F}$ is executed upon φ and this selection of sample points. It yields the following formula:

$$\mathbb{F}(\varphi,\{-1,1\}) = \begin{bmatrix} \exists x_1\ \exists x_2\ \exists x_3 \exists x_4 \\ (x_1x_4 + x_2x_4 + x_3x_2 < 0) \\ \wedge\ (x_2 > 0)\ \wedge\ (x_3 > 0)\ \wedge\ (x_4 > 0) \\ \wedge\ (x_3x_4 - x_4^2 + x_3^2 + 1 < 0) \\ \wedge\ (x_1 \geq -1)\ \wedge\ (x_1 \leq 1) \end{bmatrix}.$$

Finally, the $\exists$ **RCF** proof procedure $\mathbb{P}$ given by **RAHD**'s execution of the following proof strategy is executed upon $\mathbb{F}(\varphi, S_1)$:

```
[ simp-zrhs; run stable-simp; satur-lin;
  interval-cp(max-contractions := 30)].
```

This proof strategy is unable to reach a decision about $\mathbb{F}(\varphi, S_1)$ and returns **unknown**. This causes j to be incremented to 2, and the next step of the cell selection process is executed:

$$\mathbb{S}(S_1, 2) = \{-1\}.$$

Next, the formula construction function $\mathbb{F}$ is executed upon φ and this selection of sample points. It yields the following formula:

$$\mathbb{F}(\varphi,\{-1\}) = \begin{bmatrix} \exists x_1\ \exists x_2\ \exists x_3 \exists x_4 \\ (x_1x_4 + x_2x_4 + x_3x_2 < 0) \\ \wedge\ (x_2 > 0)\ \wedge\ (x_3 > 0)\ \wedge\ (x_4 > 0) \\ \wedge\ (x_3x_4 - x_4^2 + x_3^2 + 1 < 0) \\ \wedge\ (x_1 \geq -1)\ \wedge\ (x_1 \leq -1) \end{bmatrix}.$$

The $\exists$ **RCF** proof procedure $\mathbb{P}$ is executed upon $\mathbb{F}(\varphi, \{-1\})$, again returning **unknown**. This causes j to be incremented to 3, its current upper bound as determined by the covering width function w, and so a final cell selection will be executed upon S_1:

$$\mathbb{S}(S_1, 3) = \{1\}.$$

Next, the formula construction function $\mathbb{F}$ is executed upon φ and this selection of sample points. It yields the following formula:

$$\mathbb{F}(\varphi,\{1\}) = \left[\begin{array}{r} \exists x_1\ \exists x_2\ \exists x_3 \exists x_4 \\ (x_1x_4 + x_2x_4 + x_3x_2 < 0) \\ \wedge\ (x_2 > 0)\ \wedge\ (x_3 > 0)\ \wedge\ (x_4 > 0) \\ \wedge\ (x_3x_4 - x_4^2 + x_3^2 + 1 < 0) \\ \wedge\ (x_1 \geq 1)\ \wedge\ (x_1 \leq 1) \end{array}\right].$$

The $\exists$ **RCF** proof procedure $\mathbb{P}$ is executed upon $\mathbb{F}(\varphi,\{1\})$, and this time it is able to prove the constructed formula to be **false**. Thus, the sample point 1 can be eliminated from S_1 and we need not lift over it.

It is worth pausing and understanding why classical partial CAD was unable to eliminate 1 from S_1, yet this AP-CAD method succeeds. By inspecting the formula, we see that simple interval reasoning is enough to recognise the falsity of $\mathbb{F}(\varphi,\{1\})$, but classical partial CAD, performing only substitution, the evaluation of ground atoms, and propositional reasoning, does not recognise this.

Now, we isolate the relevant sample points induced by the univariate family $P_2[x_1 \mapsto -1]$ and continue onto the next dimension.

$(\mathbb{R}^2 \mapsto \mathbb{R}^3 \mapsto \mathbb{R}^4)$: It turns out that for the rest of the lifting process, our AP-CAD theatre is unable to eliminate any cells. This results in having to retain 20 sample points in $\mathbb{Q}^3$ and 100 sample points in $\mathbb{Q}^4$.

Thus, while this AP-CAD instance showed some promise in improving the efficiency of this example during $(\mathbb{R}^1 \mapsto \mathbb{R}^2)$ lifting, in the end it resulted in having to lift over more cells than classical partial CAD did. Let us extend our previous table so that we may compare the cardinalities of the sets of retained sample points for the three variants of lifting seen thus far:

	Normal	Partial	Intvl. AP-CAD
$\mathbb{Q}^1$	2	2	1
$\mathbb{Q}^2$	14	7	7
$\mathbb{Q}^3$	40	10	20
$\mathbb{Q}^4$	200	50	100

But, notice the following: We did not employ at all the method of classical partial CAD during this AP-CAD lifting. That is, many of these cells the AP-CAD did not

recognise to be eliminable may have been recognised to be eliminable by substitution, the evaluation of ground atoms and propositional reasoning. Let us see what happens when we combine these methods.

8.6.2.2.7 Lifting Variant IV: Classical Partial + AP-CAD In this final variant of lifting, we will first try to eliminate cells by the reasoning of classical partial CAD, and we will then apply our AP-CAD cell selection and elimination loop to the cells which survived.

($\mathbb{R}^1 \mapsto \mathbb{R}^2$): We begin with two sample points $S_1 = \{-1, 1\}$. Partial CAD elimination is unable to eliminate either of them. Our AP-CAD theatre is able to eliminate one of them, 1, as we saw before. So, we lift over -1 w.r.t. P_2 and compute 7 sample points in $\mathbb{Q}^2$

($\mathbb{R}^2 \mapsto \mathbb{R}^3$): We begin with the 7 sample points in $\mathbb{Q}^2$ and apply partial CAD elimination. This results in 2 sample points being eliminated. Our AP-CAD theatre is unable to eliminate any of them. We are then left with only having to lift over 5 points in $\mathbb{Q}^2$ w.r.t. $\mathbb{P}^3$. So, we lift over them and compute 14 sample points in $\mathbb{Q}^3$.

($\mathbb{R}^3 \mapsto \mathbb{R}^4$): Finally, we begin with the 14 sample points in $\mathbb{Q}^3$ and apply partial CAD elimination to them. This eliminates 7. We then lift over the remaining 7 points w.r.t. $\mathbb{P}_4$ and compute 35 sample points in $\mathbb{Q}^4$.

Thus, the combination of the cell elimination method of classical partial CAD coupled with our AP-CAD lifting led to the most efficient lifting variant, measured by the number of cells retained at each dimension, for this example. We may now complete our table comparing the cardinalities of the sets of sample points retained at each dimension:

	Normal	Partial	Intvl. AP-CAD	Partial + Intvl. AP-CAD
$\mathbb{Q}^1$	2	2	1	1
$\mathbb{Q}^2$	14	7	7	5
$\mathbb{Q}^3$	40	10	20	7
$\mathbb{Q}^4$	200	50	100	35

8.6.2.3 Experimental Conclusion

In this experiment, we built a concrete instance of our Abstract Partial CAD framework making use of light-weight interval arithmetic reasoning and examined its efficacy. We compared in substantial detail four variants of lifting on a particular $\exists$ **RCF** formula φ. These four methods were:

1. full-dimensional lifting without eliminating any full-dimensional cells,

2. full-dimensional lifting with standard partial CAD used to eliminate cells,

3. full-dimensional lifting with our AP-CAD theatre used to eliminate cells,

4. full-dimensional lifting with a combination of standard partial CAD and our AP-CAD theatre used to eliminate cells.

As lifting is usually the most expensive aspect of a CAD-based decision method, we focused on a comparison between the number of cells one is forced to lift over by each of these variants.

For our example formula φ, we found the final method combining classical partial CAD and our AP-CAD instance to be the best, followed by classical partial CAD, then our AP-CAD instance working alone, and finally the method of normal full-dimensional CAD without any partiality. In all cases, the cost of the AP-CAD theatre/stage execution was miniscule, measuring no more than 0.01% of the total CPU time, as the cell selection, formula construction and interval reasoning employed were each of such a simple nature.

In general, we conclude that the final variant of lifting combining classical partial CAD and our AP-CAD instance at worst performs as well as partial CAD and at best performs substantially better. This conclusion is supported by experiments we have done with other example $\exists$ **RCF** formulas. Below we summarise in table form our findings on five examples (cf. **Appendix A**) , the first being φ we worked through above:

		Normal	Partial	Intvl. AP-CAD	Partial + Intvl. AP-CAD
P1	$\mathbb{Q}^1$	2	2	1	1
	$\mathbb{Q}^2$	14	7	7	5
	$\mathbb{Q}^3$	40	10	20	7
	$\mathbb{Q}^4$	200	50	100	35
P2	$\mathbb{Q}^1$	16	8	0	0
	$\mathbb{Q}^2$	140	0	-	-
	$\mathbb{Q}^3$	664	-	-	-
P3	$\mathbb{Q}^1$	4	2	2	2
	$\mathbb{Q}^2$	20	5	10	5
	$\mathbb{Q}^3$	60	3	30	3
	$\mathbb{Q}^4$	120	6	60	6
P4	$\mathbb{Q}^1$	12	10	0	0
	$\mathbb{Q}^2$	88	19	-	-
	$\mathbb{Q}^3$	264	19	-	-
	$\mathbb{Q}^4$	1320	95	-	-
P5	$\mathbb{Q}^1$	8	3	4	3
	$\mathbb{Q}^2$	64	8	32	8
	$\mathbb{Q}^3$	512	8	56	8
	$\mathbb{Q}^4$	2560	40	1280	40

Finally, we wish to state two closing experimental observations.

First, the AP-CAD instance we constructed and experimented with in this section is but one of many (indeed, infinitely many) possible such instances. The fact that even such a simple[7] instance of the AP-CAD paradigm shows such promise is very encouraging.

Second, though we have been working solely with full-dimensional variants of CAD-based methods, AP-CAD may prove to be even more useful when it comes to full-on CAD-based methods which require irrational algebraic number computations. The reason is that through cell selection, formula construction and proof procedure execution, one has the ability to eliminate a set of *many* sample points all at once using AP-CAD, and in this way many irrational algebraic sample points may be eliminated

[7]It is worth observing that the combination of classical partial CAD and our AP-CAD instance could actually be realised by a more intricate AP-CAD instance which performs the classical partial CAD reasoning itself. In this way, AP-CAD can be seen as a true generalisation of classical partial CAD.

with only *rational* number computations. To use our concrete AP-CAD instance as an example in the context of standard CAD not restricted to full-dimensional lifting, one has the potential to eliminate a set of sample points such as $\{-3, -\sqrt{2}, -1, \sqrt{2}, \sqrt{2} + 3\sqrt{3}, 15\}$ simply by constructing and refuting a formula that only references this set of sample points using its minimal and maximal *rational* values, e.g., through a statement of the form $F \wedge (x_1 \geq -3 \ \wedge \ x_1 \leq 15)$ for some F. The ability to eliminate multiple irrational algebraic sample points simply through reasoning about formulas involving rational numbers seems very promising for the extension of these ideas to unrestricted cell decompositions.

8.7 Conclusion

In this chapter, we have presented a user-oriented view of our **RAHD** (Real Algebra in High Dimensions) proof tool for $\exists$ **RCF**. This tool can be seen as a practical realisation of many of the decision method ideas put forth in our dissertation, and provides a framework in which new $\exists$ **RCF** proof procedures may be easily built and deployed. In the process, we presented two classes of detailed experiments. Let us give a high-level summary.

First, we showed how one may use **RAHD** to build and apply a promising novel class of proof strategies combining Gröbner bases and full-dimensional cell decompositions. This class of experiments was especially nice as the automatic proof strategy we built was derived through the interactive deciding of a formula we were previously unable to decide with automatic methods. A refined variant of this proof strategy then performed very well compared to other $\exists$ **RCF** decision methods and was able to decide a number of problems which were to our knowledge previously beyond the reaches of automatic methods.

Second, we built a concrete instance of our framework of Abstract Partial Cylindrical Algebraic Decomposition introduced in **Chapter 7**. This instance utilised a simple form of interval arithmetic based reasoning as a method for short-circuiting aspects of the lifting phase of (full-dimensional) partial CAD construction. We compared the performance of this method with a number of other approaches to lifting, and ultimately came to the conclusion that classical partial CAD extended with this interval-based AP-CAD instance is the lifting method of choice for a number of problems. As this concrete instance is but one of infinitely many possible AP-CAD instances, we are excited by the promise of this general framework. We find it conceptually and prac-

tically edifying to be able to build and apply arbitrary sound but possibly incomplete $\exists$ **RCF** proof procedures in the context of a complete decision method, i.e., one based on CAD, without sacrificing the completeness of the underlying method.

Chapter 9

Conclusion

9.1 Context and Enquiry

Let us now step back from our work and reflect upon it. We will strive to place it in a larger context, expounding upon a broader vision to which it contributes.

9.1.1 What and Why?

The main goal underlying our dissertation may be stated simply: To improve our ability to decide nonlinear arithmetical conjectures over the real and complex numbers. The ubiquitous nature of these arithmetics, the natural manner in which they arise in nearly all mathematical sciences, leaves no doubt that radically improved proof procedures for them would be of remarkable utility. But, the inherent algorithmic complexity of these decision problems makes progress towards practically useful general-purpose decision engines necessarily tough going. It is safe to say that this will never change.

The story, however, need not end here. When we embrace this difficulty and recognise that no single decision method for these arithmetics will ever be sufficient, then we see there is much that can be done. There are so many different decision methods for these theories, with such a range of underlying mathematical techniques, and these methods very often disagree on which problems they find challenging. Through this immense diversity — how one method's insurmountable obstacle may be another's stroll through the garden — meaningful progress can be made.

By specialising known proof methods (and components of them) to exploit structural properties of problems of interest, and by providing mechanisms through which difficult problems can be broken down into pieces, with each piece feasibly solvable

by some available method, the decision feasibility boundary can be slowly expanded, at least for many classes of practical problems. It is easy then to come to the conclusion that developing ways in which nonlinear arithmetical decision methods may be *combined* and *specialised* is a worthy pursuit. But, where does one go from here?

From our personal experience, we found that once we developed a few such combinations, even successful ones which allowed us to solve problems we could not solve before, an unpleasant feeling began to take hold. Were we just building an ad hoc "bag of tricks?" Were our decision method combinations just a "bunch of hacks?" What is the *science* underlying these heuristic combinations? What knowledge can be gained from them? To do good science, one must strive to find and understand the underlying *principles*. One feels this as a need, like water, oxygen, time spent with nature. How then can we undertake this work, to build these combined proof procedures which are effective in large part *because of their bags of tricks,* without betraying our conscience? Can we do this in a principled way? To us, our most pleasing contributions have been attempted answers to these questions.

9.1.2 A Search for Underlying Principles

There are three main principled approaches we have given for specialising and combining nonlinear proof procedures over the real and complex numbers.

Abstract Gröbner Bases In this theory, specialised Gröbner basis construction algorithms became formal *strategies* for sequencing a small set of inference rules (cf. **Chapter 3**). These formal strategies were much easier to reason about than the actual algorithms to which they corresponded. With this framework, we were able to prove the correctness of novel Gröbner basis algorithms effective on a new class of problems arising in large-scale SMT-based program verification (cf. **Chapter 5**). In addition, we were able to prove that certain classical Gröbner basis optimisation methods, so-called "superfluous S-polynomial criteria," could be soundly exploited in the context of any correct formal strategy. This allowed us to include these optimisations in our algorithms, which contributed greatly to their practical success. The underlying *principle* of this work comes to light through the use of an abstract framework for building and analysing our specialised decision methods. In contrast to the next piece of work we describe, we consider this principled approach to be *global*, as the entire Gröbner basis decision method was abstracted and made manipulable and analysable by the

framework.

Abstract Partial CAD With this work, we realised that the ideas underlying the partial CAD method of Collins and Hong could be generalised to work with *arbitrary* sound but possibly incomplete $\exists$ **RCF** proof procedures (cf. **Chapter 7**). This generalisation allowed one to build custom decision methods in which algorithmic data describing how one eliminates cells during CAD construction could be given as parameters to a new higher-order CAD-based proof procedure. This facilitates the injection of *strategy* into a key aspect of CAD construction. The *principle* underlying this work is two-fold: First, it provides a general setting in which a broad class of modifications to CAD lifting can all be seen to be special cases of a single idea. Second, it provides a way in which custom "ad hoc" sound but possibly incomplete combinations of proof procedures can be exploited in the context of a *complete* procedure **without** sacrificing this completeness. In contrast to the work involving Abstract Gröbner Bases, we consider the first principle underlying this work to be a *local* one. This is because only one component of the partial CAD procedure was abstracted and made strategically controllable. To realise this general idea and actually bring it to bear on problems, we had to have a method for building and deploying custom $\exists$ **RCF** proof procedures as first-class objects. This is taken up with the next contribution we describe.

RAHD Strategies Finally, we come to our $\exists$ **RCF** proof tool **RAHD** (cf. **Chapter 8**). With this work, we built a tool in which a large heterogeneous collection of nonlinear arithmetical proof procedures could be effectively combined and applied. Through tailoring these combinations, proof procedures specialised to exploit structural properties of problems of interest could be synthesised. There were a few key underlying *principles* allowing this to work: First, we had to build a large collection of $\exists$ **RCF** proof procedures sharing a common interface, with many of them allowing strategic control to be exerted over them through the use of parameters (cf. **Chapter 6**). Second, we built a simple *proof strategy* language for expressing conditional combinations of these proof procedures. In doing so, radically different proof procedure combinations were placed on the same footing as each simply being different **RAHD** proof strategies. Third, these strategies were made first-class objects in the sense that they could be passed around to each other as functional parameters. This allowed us to actually build

the Abstract Partial CAD framework and experiment with it. Fourth, the system was built with an intended methodology, roughly as follows: When faced with a difficult problem none of the known proof strategies can solve, one uses interactive methods to investigate the formula and try to synthesise an applicable proof procedure. If one succeeds, then this proof procedure can be made available as a push-button automatic method, so that the resulting technique fits cleanly into formal verification tool-chains.

Overall, these ideas give firm foundations upon which specialised nonlinear arithmetical proof procedures may be built and deployed in a principled way. Many of the ideas have been already realised in our tool **RAHD** and in the SMT solver Z3, and are showing much practical promise. These results culminate in a novel contribution to our overall goal of improving our ability to decide nonlinear arithmetical conjectures over the real and complex numbers.

9.2 Future Work

Proposals for future work have been woven throughout this thesis. Let us now end with a slightly expanded summary of a few of the future directions we most hope to pursue.

Abstract Gröbner Bases

First, it would be very interesting to investigate the strategy-independent admissibility of additional superfluous S-polynomial critera w.r.t. Abstract GBs. We were able to prove the admissibility of three such criteria, but there are many widely used criteria which we have not begun to analyse. One first place to start (suggested to us by James Davenport [Dav09]) is to see if our proof of the strategy-independent admissibility of **Criterion 2** could be adapted to a proof of the strategy-independent admissibility of the so-called "chained Buchberger-2" criterion [Buc79].

In similar spirit, we would also like to see if any of the linear algebraic techniques utilised by Faugere's F4 and F5 Gröbner basis construction algorithms could be brought to bear on our algorithms, so that the combined methods could be even more effective on classes of large, largely linear ("L3") nonlinear systems [Fau99, Fau02]. If we are able to do this, then based upon the experiments in **Section 5.3**, it also seems plausible that the effectivity of our algorithms could be extended to large systems with a higher nonlinear component than the L3 ones currently amenable to our methods.

Finally, it could also be very rewarding to build a tool, perhaps somewhat similar to **RAHD**, in which new Gröbner basis algorithms could be synthesised through their expression in a strategy language based upon the Abstract GB calculus.

Abstract Partial CAD

Recall that Abstract Partial CAD as developed in this thesis only allows one to exert strategic control over the lifting phase of CAD. As described at the end of **Chapter 8**, it would be very interesting to work on extending the AP-CAD framework to include methods for exerting strategic control also over the projection phase of partial CAD construction. We sketched one high-level idea for how we might go about this based upon some recent advances in the algorithmic construction of sums of squares decompositions. This would be both conceptually edifying and potentially very useful in practice, as the size of projection (factor) sets can grow astronomically as one approaches $\mathbb{Z}[x_1]$ from above.

In **Section 8.6.2**, we built a concrete instance of AP-CAD and analysed in detail its lifting performance on a collection of problems. This instance used an AP-CAD theatre based on light-weight interval constraint propagation to recognise when certain partial CAD cells could be eliminated during lifting. These experiments suggest that the AP-CAD approach may be of much practical use. Thus, we wish to develop many more robust AP-CAD stages and theatres tailored to difficult $\exists$ **RCF** problem classes. This will require much experimentation and tool support, and will likely drive improvements to **RAHD** with new techniques for aiding the construction and analysis of AP-CAD instances.

On the immediate horizon, we are especially interested in building and applying AP-CAD instances which make use of quadratic virtual term substitution [Wei97] and the use of semidefinite programming to find Positivstellensatz witnesses [Par03, Har07]. We would also like to further extend our implementation to handle the case of general (i.e., not just full-dimensional) cell decompositions. To do this, we will need to build into **RAHD** native support for computing with irrational real algebraic numbers.

RAHD

Let us turn our focus now to **RAHD** itself. There are many ways the system can be improved.

First, we would like to include more **RCF** (semi-)decision methods as primitive proof procedures in the system. As with those we have built into **RAHD** so far, we

expect that through the process of studying, implementing and experimenting with these additional proof methods, we may find ways to further generalise and parameterise them so that they may be combined with other **RAHD** CMFs and proof strategies in compelling ways. Looking forward, the first five such procedures we hope to work on next are quadratic virtual term substitution[1] [Wei97], Basu-Pollack-Roy connected component sampling [BPR06], Positivstellensatzen search using semidefinite programming [Har07], more advanced ICP-based methods [Rat06] and a new technique for computing CADs based on complex triangulation [CMXY09].

Next, the **RAHD** proof strategy language is currently extremely simple. It would be very interesting to extend the language with more constructs, especially with some form of parallelism enabling simultaneous proof strategy execution with back-tracking.

Throughout our experiments presented in **Section 8.6**, we found the need to add much proof procedure tracing and profiling machinery to the system. This is how we were able to auto-generate the tables found in **Appendix B**, for instance. In working to refine proof strategies for problem corpora, we found having access to this sort of profiling data extremely useful. But, there are many ways we can imagine improving this type of support in the system with more fine-grained performance analyses. Probably, we will over time improve this aspect of the system through undertaking more and more case studies which then push us to add additional tool support as it is needed.

Finally, we would like to apply **RAHD** to a much broader set of problem families. In the process, we hope to both develop more novel, practically useful proof strategies and apply them in the context of many more serious proof efforts. This will undoubtedly drive serious improvements to the tool. We also expect it to lead to interesting theoretical developments, many of which may result in practically useful fruit. Fortunately, at the time of this writing, we have been given a four-year EPSRC grant to continue this work [JPP10], with our key foci being the application of **RAHD** within (i) the formal verification of a class of mixed discrete-continuous dynamical system known as *hybrid systems*, including the integration of **RAHD** with Paulson's MetiTarski [AP10] prover for **RCF** extended with special (trigonometric, exponential) functions, (ii) the formal verification of hardware, software and bioware through the integration of techniques derived from **RAHD** within SMT solvers, and (iii) formalised mathematics.

[1] At the time of this writing, we have in fact implemented basic quadratic virtual term substitution (VTS) already into the latest version of **RAHD**. But, as intermediate formulas produced during VTS can be very large, we need to implement more of the simplification methods found in Dolzmann's dissertation [Dol00] before our VTS implementation in **RAHD** is ready for serious use.

Towards this first application aim, we are planning to integrate **RAHD** as the nonlinear real arithmetical back-end of a number of hybrid systems verification tools including SRI's HybridSAL and CMU's KeYmaera. Our initial focus has been the integration of **RAHD** as the **RCF** backend of Paulson's MetiTarski, which has already been completed. During this integration, we have constructed a small collection of **RAHD** strategies tailored to MetiTarski problems, and the application of these strategies within MetiTarski has allowed the system to prove theorems which were previously beyond its reach. There is much work to be done to obtain a robust integration of the two tools.

Towards the second aim above focused on SMT solvers, we have ongoing collaboration with de Moura in which we are devising more nonlinear arithmetical methods which can be effectively integrated within Z3. This extends our work on Abstract GBs and combined **RCF** proof procedures in **RAHD**, and **RAHD** is being used to develop the first prototype versions of these new SMT-aimed nonlinear arithmetic proof techniques. In the process, we have also been working together on designing an explicit strategy language for Z3 and other SMT solvers, and de Moura is now building the latest 2011 competition version of Z3 based upon these new strategy mechanisms [dMP11].

Appendix A

Obtaining RAHD and Supporting Documents

Our proof tool **RAHD**, its source code, the verification of the relevant portions of its generalised interval arithmetic machinery, formulas used in our experiments and other related documents may be obtained from our

Thesis Supporting Data URL

at

`http://homepages.inf.ed.ac.uk/s0793114/phd-thesis/`.

Appendix B

Fine-Grained Data on Calculemus Strategy Execution

In this appendix, we give more detailed data accounting for the performance differences between the three calculemus strategies reported on in **Chapter 8**. For a discussion of this data, please refer to **Section 8.6.1.2**.

For each considered problem P, and each calculemus strategy S, we report:

- The total number of cases in the proof tree constructed by S in its solution of P,
- For each CMF F applied by S during its deciding of P, we give the number of times S *successfully* applied F, and the total amount of time S spent *attempting* to apply F. Note that when S attempts to apply F, this application may either succeed (make progress on a case), or fail (not make progress on a case).

  We report this data in a table. In each table, the CMF contribution column for strategy S has entries of the form "N (T)." N is the number of times S applied F successfully and T is the total amount of time spent attempting to apply F (in seconds). If a given strategy does not even attempt to apply a CMF listed, then the corresponding CMF contribution entry consists of a dash.

  The final row of the table for P lists the total time each S took in solving P.

Note that the timing reported below will differ (quantitatively, not qualitatively) from that reported in **Figure 8.6.1.2**. This is because recording this proof profiling data has a small overhead. This profiling was not enabled during the experiments reported on in the main text.

Note also that in each table, the total time each strategy S took in solving problem P may be a bit larger than the sum of the CMF execution times given for S. This is because there are other computationally non-trivial aspects of **RAHD** execution besides the application of CMFs. This includes various system book-keeping mechanics, the maintenance of a number of caches, the evaluation of measure-values upon cases, garbage collection in between CMF applications, and so on.

P0	calc-0	calc-1	calc-2
#(Proof-tree)	1024	1024	1024
QEPCAD (OPEN?:=1)	128 (0.754)	89 (0.619)	89 (0.621)
RCR-INEQS	0 (0.000)	0 (0.000)	0 (0.001)
INTERVAL-CP	-	0 (0.151)	0 (0.151)
UNIV-STURM-INEQS	60 (0.006)	17 (0.004)	17 (0.004)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	210 (0.024)	107 (0.018)	107 (0.017)
DEMOD-NUM	1023 (0.056)	107 (0.005)	107 (0.004)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.076)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	916 (0.664)	916 (0.702)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.009)	1 (0.005)	1 (0.006)
SIMP-ZRHS	0 (0.001)	0 (0.000)	0 (0.000)
SIMP-GLS	1023 (0.022)	107 (0.003)	107 (0.003)
DEMOD-LIN	0 (0.001)	0 (0.001)	0 (0.002)
FERT-TSOS	23 (0.009)	2 (0.006)	2 (0.006)
SATUR-LIN	128 (0.042)	89 (0.033)	89 (0.034)
TRIV-IDEALS	0 (0.001)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	0.963	1.627	1.729

P1	calc-0	calc-1	calc-2
#(Proof-tree)	3072	3072	3072
QEPCAD (OPEN?:=1)	378 (1.203)	194 (0.806)	194 (0.994)
RCR-INEQS	0 (0.000)	0 (0.000)	0 (0.001)
INTERVAL-CP	-	112 (0.421)	0 (0.316)
UNIV-STURM-INEQS	156 (0.023)	83 (0.010)	13 (0.003)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	841 (0.085)	546 (0.045)	294 (0.045)
DEMOD-NUM	3072 (0.148)	486 (0.021)	207 (0.015)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	279 (0.308)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	2586 (1.569)	2586 (1.570)
SPLIT-INEQS (MAX-SPLITS:=12)	3 (0.023)	3 (0.035)	3 (0.017)
SIMP-ZRHS	0 (0.002)	0 (0.000)	0 (0.001)
SIMP-GLS	3072 (0.059)	486 (0.020)	207 (0.003)
DEMOD-LIN	0 (0.004)	0 (0.005)	0 (0.001)
FERT-TSOS	99 (0.029)	15 (0.016)	0 (0.016)
SATUR-LIN	378 (0.088)	306 (0.077)	194 (0.056)
TRIV-IDEALS	0 (0.000)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	1.822	3.261	3.601

P2	calc-0	calc-1	calc-2
#(Proof-tree)	768	768	768
QEPCAD (OPEN?:=1)	99 (0.823)	99 (0.894)	99 (0.820)
RCR-INEQS	0 (0.000)	0 (0.001)	0 (0.000)
INTERVAL-CP	-	0 (0.326)	0 (0.333)
UNIV-STURM-INEQS	96 (0.017)	72 (0.019)	72 (0.014)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	165 (0.167)	143 (0.137)	143 (0.140)
DEMOD-NUM	768 (0.051)	171 (0.010)	171 (0.008)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.189)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	597 (0.880)	597 (0.904)
SPLIT-INEQS (MAX-SPLITS:=12)	3 (0.003)	3 (0.002)	3 (0.002)
SIMP-ZRHS	0 (0.001)	0 (0.000)	0 (0.000)
SIMP-GLS	768 (0.061)	171 (0.003)	171 (0.009)
DEMOD-LIN	0 (0.005)	0 (0.006)	0 (0.005)
FERT-TSOS	0 (0.019)	0 (0.018)	0 (0.020)
SATUR-LIN	99 (0.054)	99 (0.057)	99 (0.058)
TRIV-IDEALS	0 (0.000)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	1.292	2.475	2.626

P3	calc-0	calc-1	calc-2
#(Proof-tree)	768	768	768
QEPCAD (OPEN?:=1)	99 (0.978)	99 (1.020)	99 (1.041)
RCR-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	0 (0.358)	0 (0.330)
UNIV-STURM-INEQS	96 (0.026)	72 (0.016)	72 (0.015)
SIMP-REAL-NULL	0 (0.000)	0 (0.001)	0 (0.000)
SIMP-ARITH	165 (0.166)	143 (0.149)	143 (0.140)
DEMOD-NUM	768 (0.051)	171 (0.008)	171 (0.009)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.185)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	597 (0.896)	597 (0.885)
SPLIT-INEQS (MAX-SPLITS:=12)	3 (0.002)	3 (0.002)	3 (0.003)
SIMP-ZRHS	768 (0.076)	171 (0.016)	171 (0.018)
SIMP-GLS	768 (0.035)	171 (0.008)	171 (0.008)
DEMOD-LIN	0 (0.005)	0 (0.005)	0 (0.005)
FERT-TSOS	0 (0.037)	0 (0.016)	0 (0.017)
SATUR-LIN	99 (0.053)	99 (0.058)	99 (0.056)
TRIV-IDEALS	0 (0.000)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.001)	0 (0.000)	0 (0.000)
	1.520	2.665	2.844

P4	calc-0	calc-1	calc-2
#(Proof-tree)	768	768	768
QEPCAD (OPEN?:=1)	99 (0.724)	90 (0.686)	90 (0.688)
RCR-INEQS	0 (0.000)	0 (0.000)	0 (0.001)
INTERVAL-CP	-	0 (0.205)	0 (0.213)
UNIV-STURM-INEQS	88 (0.010)	50 (0.009)	50 (0.009)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	194 (0.062)	139 (0.050)	139 (0.050)
DEMOD-NUM	768 (0.056)	140 (0.008)	140 (0.006)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.157)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	628 (0.890)	628 (0.895)
SPLIT-INEQS (MAX-SPLITS:=12)	3 (0.010)	3 (0.003)	3 (0.002)
SIMP-ZRHS	0 (0.002)	0 (0.000)	0 (0.000)
SIMP-GLS	768 (0.032)	140 (0.004)	140 (0.005)
DEMOD-LIN	0 (0.003)	0 (0.002)	0 (0.002)
FERT-TSOS	8 (0.012)	0 (0.012)	0 (0.011)
SATUR-LIN	99 (0.037)	90 (0.041)	90 (0.036)
TRIV-IDEALS	0 (0.000)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	1.035	2.013	2.188

P5	calc-0	calc-1	calc-2
#(Proof-tree)	4	4	4
TRIV-IDEALS	4 (0.171)	4 (0.154)	4 (0.154)
SATUR-LIN	4 (0.023)	4 (0.033)	4 (0.037)
FERT-TSOS	0 (0.002)	0 (0.002)	0 (0.001)
DEMOD-LIN	0 (0.004)	0 (0.004)	0 (0.005)
SIMP-GLS	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ZRHS	4 (0.002)	4 (0.002)	4 (0.002)
SPLIT-INEQS (MAX-SPLITS:=12)	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	0 (0.003)	0 (0.003)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.005)
DEMOD-NUM	0 (0.000)	0 (0.002)	0 (0.001)
SIMP-ARITH	0 (0.002)	0 (0.002)	0 (0.002)
SIMP-REAL-NULL	0 (0.002)	0 (0.002)	0 (0.003)
UNIV-STURM-INEQS	0 (0.000)	0 (0.001)	0 (0.001)
INTERVAL-CP	-	0 (0.007)	0 (0.007)
	0.257	0.263	0.273

P6	calc-0	calc-1	calc-2
#(Proof-tree)	8	8	1
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	8 (0.003)	1 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	-
QEPCAD (OPEN?:=1)	0 (0.039)	-	-
RCR-INEQS	6 (0.675)	-	-
SATUR-LIN	8 (0.008)	-	-
FERT-TSOS	0 (0.001)	-	-
DEMOD-LIN	8 (0.001)	-	-
SIMP-GLS	17 (0.001)	-	-
SIMP-ZRHS	14 (0.000)	-	-
DEMOD-NUM	12 (0.002)	-	-
SIMP-ARITH	12 (0.000)	-	-
SIMP-REAL-NULL	0 (0.000)	-	-
UNIV-STURM-INEQS	0 (0.000)	-	-
TRIV-IDEALS	0 (0.731)	-	-
INT-DOM-ZPB	0 (0.001)	-	-
QEPCAD	6 (146.511)	-	-
	148.026	0.054	0.001

P7	calc-0	calc-1	calc-2
#(Proof-tree)	1	1	1
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	1 (0.000)	1 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	0 (0.000)	0 (0.000)	-
DEMOD-LIN	0 (0.000)	-	-
SIMP-GLS	0 (0.000)	-	-
SIMP-ZRHS	1 (0.000)	-	-
DEMOD-NUM	0 (0.000)	-	-
SIMP-ARITH	0 (0.001)	-	-
SIMP-REAL-NULL	1 (0.000)	-	-
	0.040	0.001	0.000

P8	calc-0	calc-1	calc-2
#(Proof-tree)	64	64	1
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	64 (0.089)	1 (0.001)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.001)	1 (0.000)	-
QEPCAD (OPEN?:=1)	24 (0.075)	-	-
RCR-INEQS	48 (1.503)	-	-
SATUR-LIN	64 (0.029)	-	-
FERT-TSOS	0 (0.008)	-	-
DEMOD-LIN	62 (0.011)	-	-
SIMP-GLS	52 (0.002)	-	-
SIMP-ZRHS	100 (0.003)	-	-
DEMOD-NUM	42 (0.005)	-	-
SIMP-ARITH	0 (0.017)	-	-
SIMP-REAL-NULL	0 (0.002)	-	-
UNIV-STURM-INEQS	0 (0.002)	-	-
TRIV-IDEALS	16 (2.570)	-	-
INT-DOM-ZPB	0 (0.002)	-	-
QEPCAD	24 (0.148)	-	-
	4.396	0.142	0.048

P9	calc-0	calc-1	calc-2
#(Proof-tree)	128	128	2
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	128 (0.089)	2 (0.001)
SPLIT-INEQS (MAX-SPLITS:=12)	2 (0.001)	2 (0.001)	-
QEPCAD (OPEN?:=1)	56 (0.165)	-	-
RCR-INEQS	96 (1.288)	-	-
SATUR-LIN	128 (0.047)	-	-
FERT-TSOS	0 (0.013)	-	-
DEMOD-LIN	124 (0.020)	-	-
SIMP-GLS	200 (0.010)	-	-
SIMP-ZRHS	190 (0.003)	-	-
DEMOD-NUM	148 (0.004)	-	-
SIMP-ARITH	0 (0.060)	-	-
SIMP-REAL-NULL	0 (0.003)	-	-
UNIV-STURM-INEQS	0 (0.003)	-	-
TRIV-IDEALS	32 (2.545)	-	-
INT-DOM-ZPB	0 (0.002)	-	-
QEPCAD	24 (0.154)	-	-
	4.343	0.145	0.048

P10	calc-0	calc-1	calc-2
#(Proof-tree)	32768	8192	8192
TRIV-IDEALS	278 (45.847)	96 (0.936)	21 (0.101)
SATUR-LIN	485 (1.465)	344 (0.756)	25 (0.275)
FERT-TSOS	70 (0.355)	48 (0.200)	8 (0.041)
DEMOD-LIN	1101 (0.238)	444 (0.048)	33 (0.003)
SIMP-GLS	34351 (1.290)	3384 (0.185)	33 (0.000)
SIMP-ZRHS	32919 (0.370)	3133 (0.031)	33 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	8194 (0.468)	2 (0.109)	2 (0.131)
QEPCAD (OPEN?:=1)	207 (21.081)	70 (10.881)	-
RCR-INEQS	180 (7.601)	70 (0.225)	-
INT-DOM-ZPB	0 (0.012)	0 (0.006)	-
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	5120 (6.052)	5120 (6.092)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	3039 (2.126)
DEMOD-NUM	32766 (3.318)	3070 (0.170)	33 (0.003)
SIMP-ARITH	1954 (0.430)	482 (0.143)	33 (0.006)
SIMP-REAL-NULL	142 (0.031)	0 (0.032)	0 (0.002)
UNIV-STURM-INEQS	4 (0.035)	0 (0.016)	0 (0.001)
INTERVAL-CP	-	178 (0.834)	4 (0.024)
	100.850	21.049	9.343

P11	calc-0	calc-1	calc-2
#(Proof-tree)	32	32	20
QEPCAD (OPEN?:=1)	32 (0.072)	6 (0.015)	6 (0.016)
RCR-INEQS	16 (0.002)	4 (0.001)	4 (0.000)
INTERVAL-CP	-	2 (0.049)	2 (0.026)
UNIV-STURM-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	0 (0.002)	0 (0.000)	0 (0.000)
DEMOD-NUM	0 (0.001)	0 (0.000)	0 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.005)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	24 (0.012)	12 (0.008)
SPLIT-INEQS (MAX-SPLITS:=12)	16 (0.000)	16 (0.000)	4 (0.000)
SIMP-ZRHS	16 (0.000)	4 (0.000)	4 (0.000)
SIMP-GLS	0 (0.008)	0 (0.002)	0 (0.000)
DEMOD-LIN	16 (0.000)	4 (0.001)	4 (0.000)
FERT-TSOS	0 (0.004)	0 (0.001)	0 (0.001)
SATUR-LIN	32 (0.015)	8 (0.004)	8 (0.004)
TRIV-IDEALS	0 (1.476)	0 (0.374)	0 (0.365)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	1.688	0.539	0.502

P12	calc-0	calc-1	calc-2
#(Proof-tree)	16	16	16
QEPCAD	2 (0.001)	2 (0.002)	2 (0.002)
INT-DOM-ZPB	2 (0.001)	0 (0.000)	0 (0.000)
TRIV-IDEALS	0 (0.464)	0 (0.273)	0 (0.278)
SATUR-LIN	5 (0.001)	3 (0.000)	3 (0.001)
FERT-TSOS	1 (0.000)	0 (0.000)	0 (0.001)
DEMOD-LIN	12 (0.001)	3 (0.000)	3 (0.000)
SIMP-GLS	31 (0.000)	5 (0.003)	5 (0.000)
SIMP-ZRHS	16 (0.000)	3 (0.000)	3 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	1 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	13 (0.009)	13 (0.004)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.001)
DEMOD-NUM	17 (0.001)	2 (0.000)	2 (0.000)
SIMP-ARITH	15 (0.000)	2 (0.000)	2 (0.000)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
UNIV-STURM-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	0 (0.017)	0 (0.002)
RCR-INEQS	0 (0.272)	0 (0.002)	0 (0.000)
QEPCAD (OPEN?:=1)	1 (0.004)	1 (0.003)	1 (0.007)
	0.758	0.314	0.352

P13	calc-0	calc-1	calc-2
#(Proof-tree)	256	256	256
QEPCAD (OPEN?:=1)	22 (3.781)	21 (3.714)	21 (3.769)
RCR-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	0 (0.039)	0 (0.036)
UNIV-STURM-INEQS	6 (0.003)	4 (0.002)	4 (0.002)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	64 (0.002)	32 (0.004)	32 (0.002)
DEMOD-NUM	255 (0.009)	32 (0.000)	32 (0.002)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.021)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	223 (0.150)	223 (0.193)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.002)	1 (0.001)	1 (0.001)
SIMP-ZRHS	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-GLS	278 (0.007)	32 (0.001)	32 (0.002)
DEMOD-LIN	0 (0.001)	0 (0.001)	0 (0.000)
FERT-TSOS	14 (0.005)	8 (0.004)	8 (0.005)
SATUR-LIN	22 (0.014)	21 (0.011)	21 (0.012)
TRIV-IDEALS	0 (0.000)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	3.841	3.990	4.111

P14	calc-0	calc-1	calc-2
#(Proof-tree)	256	256	1
QEPCAD	13 (0.022)	8 (0.018)	1 (0.004)
INT-DOM-ZPB	0 (0.001)	0 (0.001)	-
TRIV-IDEALS	25 (4.169)	5 (1.337)	-
SATUR-LIN	50 (0.008)	16 (0.004)	-
FERT-TSOS	0 (0.003)	0 (0.001)	-
DEMOD-LIN	0 (0.003)	0 (0.001)	-
SIMP-GLS	256 (0.004)	4 (0.001)	-
SIMP-ZRHS	257 (0.007)	17 (0.001)	-
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.001)	1 (0.001)	-
DEMOD-NUM	248 (0.011)	5 (0.000)	-
SIMP-ARITH	64 (0.004)	0 (0.002)	-
SIMP-REAL-NULL	0 (0.001)	0 (0.000)	-
UNIV-STURM-INEQS	30 (0.003)	0 (0.000)	-
INTERVAL-CP	-	0 (0.035)	-
RCR-INEQS	22 (0.004)	10 (0.001)	-
QEPCAD (OPEN?:=1)	5 (0.042)	3 (0.012)	-
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	240 (0.093)	0 (0.001)
	4.351	1.576	0.053

P15	calc-0	calc-1	calc-2
#(Proof-tree)	8	8	8
QEPCAD	8 (0.008)	8 (0.007)	8 (0.007)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
TRIV-IDEALS	0 (0.096)	0 (0.092)	0 (0.090)
SATUR-LIN	8 (0.001)	8 (0.002)	8 (0.001)
FERT-TSOS	0 (0.001)	0 (0.004)	0 (0.000)
DEMOD-LIN	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-GLS	0 (0.001)	0 (0.000)	0 (0.000)
SIMP-ZRHS	8 (0.000)	8 (0.001)	8 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	0 (0.000)	0 (0.001)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.001)
DEMOD-NUM	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	0 (0.000)	0 (0.000)	0 (0.001)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
UNIV-STURM-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	0 (0.005)	0 (0.007)
RCR-INEQS	0 (0.000)	0 (0.001)	0 (0.000)
QEPCAD (OPEN?:=1)	0 (0.007)	0 (0.004)	0 (0.008)
	0.116	0.121	0.122

P16	calc-0	calc-1	calc-2
#(Proof-tree)	131	128	128
QEPCAD	10 (0.352)	7 (0.098)	7 (0.097)
INT-DOM-ZPB	3 (0.002)	0 (0.000)	0 (0.000)
TRIV-IDEALS	4 (4.626)	0 (1.721)	0 (1.714)
SATUR-LIN	76 (0.017)	44 (0.008)	44 (0.008)
FERT-TSOS	3 (0.005)	1 (0.002)	1 (0.002)
DEMOD-LIN	64 (0.014)	36 (0.003)	36 (0.002)
SIMP-GLS	161 (0.002)	67 (0.007)	67 (0.001)
SIMP-ZRHS	130 (0.002)	64 (0.001)	64 (0.008)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	1 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	64 (0.035)	64 (0.032)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.017)
DEMOD-NUM	120 (0.003)	48 (0.000)	48 (0.001)
SIMP-ARITH	96 (0.009)	48 (0.003)	48 (0.004)
SIMP-REAL-NULL	1 (0.001)	1 (0.001)	1 (0.000)
UNIV-STURM-INEQS	0 (0.001)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	19 (0.069)	19 (0.088)
RCR-INEQS	54 (3.259)	19 (0.002)	19 (0.003)
QEPCAD (OPEN?:=1)	49 (0.121)	17 (0.075)	17 (0.047)
	8.446	2.098	2.107

P17	calc-0	calc-1	calc-2
#(Proof-tree)	6	6	6
QEPCAD	1 (0.180)	1 (0.131)	1 (0.131)
INT-DOM-ZPB	2 (0.000)	2 (0.000)	2 (0.000)
TRIV-IDEALS	0 (0.277)	0 (0.272)	0 (0.275)
SATUR-LIN	3 (0.001)	3 (0.001)	3 (0.004)
FERT-TSOS	0 (0.000)	0 (0.000)	0 (0.000)
DEMOD-LIN	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-GLS	10 (0.001)	2 (0.001)	2 (0.000)
SIMP-ZRHS	4 (0.000)	3 (0.000)	3 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	1 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	5 (0.002)	5 (0.002)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.001)
DEMOD-NUM	7 (0.000)	2 (0.000)	2 (0.001)
SIMP-ARITH	5 (0.000)	2 (0.000)	2 (0.001)
SIMP-REAL-NULL	1 (0.000)	0 (0.000)	0 (0.000)
UNIV-STURM-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	0 (0.002)	0 (0.017)
RCR-INEQS	3 (0.094)	3 (0.092)	3 (0.095)
QEPCAD (OPEN?:=1)	0 (0.002)	0 (0.002)	0 (0.002)
	0.613	0.516	0.540

P18	calc-0	calc-1	calc-2
#(Proof-tree)	16	16	16
QEPCAD	5 (0.062)	5 (0.017)	5 (0.012)
INT-DOM-ZPB	3 (0.000)	3 (0.000)	3 (0.000)
TRIV-IDEALS	0 (0.716)	0 (0.714)	0 (0.725)
SATUR-LIN	8 (0.002)	8 (0.002)	8 (0.002)
FERT-TSOS	7 (0.001)	3 (0.000)	3 (0.004)
DEMOD-LIN	8 (0.001)	6 (0.000)	6 (0.001)
SIMP-GLS	14 (0.000)	4 (0.000)	4 (0.001)
SIMP-ZRHS	0 (0.000)	0 (0.000)	0 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	1 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	7 (0.006)	7 (0.005)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.013)
DEMOD-NUM	11 (0.001)	4 (0.000)	4 (0.001)
SIMP-ARITH	11 (0.002)	4 (0.001)	4 (0.000)
SIMP-REAL-NULL	0 (0.000)	0 (0.001)	0 (0.000)
UNIV-STURM-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	0 (0.014)	0 (0.012)
RCR-INEQS	7 (0.452)	7 (0.452)	7 (0.446)
QEPCAD (OPEN?:=1)	1 (0.013)	1 (0.009)	1 (0.012)
	1.306	1.225	1.249

P19	calc-0	calc-1	calc-2
#(Proof-tree)	256	256	256
QEPCAD (OPEN?:=1)	256 (1.531)	25 (0.095)	25 (0.101)
RCR-INEQS	0 (0.001)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	75 (0.499)	75 (0.498)
UNIV-STURM-INEQS	0 (0.034)	0 (0.015)	0 (0.016)
SIMP-REAL-NULL	0 (0.000)	0 (0.001)	0 (0.001)
SIMP-ARITH	256 (0.399)	100 (0.105)	100 (0.097)
DEMOD-NUM	0 (0.003)	0 (0.000)	0 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.225)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	156 (0.566)	156 (0.559)
SPLIT-INEQS (MAX-SPLITS:=12)	0 (0.001)	0 (0.000)	0 (0.000)
SIMP-ZRHS	256 (0.189)	100 (0.077)	100 (0.076)
SIMP-GLS	256 (0.009)	100 (0.003)	100 (0.004)
DEMOD-LIN	0 (0.036)	0 (0.015)	0 (0.015)
FERT-TSOS	0 (0.248)	0 (0.047)	0 (0.070)
SATUR-LIN	256 (0.513)	100 (0.187)	100 (0.185)
TRIV-IDEALS	0 (0.001)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	3.063	1.694	2.027

P20	calc-0	calc-1	calc-2
#(Proof-tree)	16	16	16
QEPCAD	6 (0.007)	5 (0.007)	5 (0.006)
INT-DOM-ZPB	0 (0.000)	0 (0.001)	0 (0.000)
TRIV-IDEALS	0 (0.984)	0 (0.446)	0 (1.127)
SATUR-LIN	11 (0.002)	7 (0.001)	7 (0.002)
FERT-TSOS	0 (0.000)	0 (0.000)	0 (0.000)
DEMOD-LIN	5 (0.001)	3 (0.001)	3 (0.000)
SIMP-GLS	19 (0.000)	7 (0.001)	7 (0.001)
SIMP-ZRHS	20 (0.000)	10 (0.000)	10 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	1 (0.000)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	8 (0.005)	8 (0.005)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.002)
DEMOD-NUM	15 (0.001)	6 (0.000)	6 (0.001)
SIMP-ARITH	12 (0.000)	6 (0.000)	6 (0.000)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
UNIV-STURM-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	2 (0.021)	2 (0.010)
RCR-INEQS	8 (0.091)	3 (0.255)	3 (0.402)
QEPCAD (OPEN?:=1)	3 (0.010)	0 (0.006)	0 (0.008)
	1.148	0.750	1.619

P21	calc-0	calc-1	calc-2
#(Proof-tree)	64	64	1
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	64 (0.024)	1 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	-
INT-DOM-ZPB	0 (0.000)	-	-
TRIV-IDEALS	0 (0.000)	-	-
UNIV-STURM-INEQS	12 (0.002)	-	-
SIMP-REAL-NULL	0 (0.000)	-	-
SIMP-ARITH	0 (0.000)	-	-
DEMOD-NUM	63 (0.001)	-	-
SIMP-ZRHS	64 (0.000)	-	-
SIMP-GLS	63 (0.001)	-	-
DEMOD-LIN	0 (0.001)	-	-
FERT-TSOS	0 (0.001)	-	-
SATUR-LIN	7 (0.002)	-	-
RCR-INEQS	0 (0.000)	-	-
QEPCAD (OPEN?:=1)	7 (0.010)	-	-
	0.024	0.025	0.000

P22	calc-0	calc-1	calc-2
#(Proof-tree)	2	2	2
QEPCAD (OPEN?:=1)	1 (0.001)	1 (0.001)	1 (0.001)
RCR-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
INTERVAL-CP	-	0 (0.001)	0 (0.001)
UNIV-STURM-INEQS	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-REAL-NULL	0 (0.000)	0 (0.000)	0 (0.000)
SIMP-ARITH	0 (0.000)	0 (0.000)	0 (0.000)
DEMOD-NUM	0 (0.000)	0 (0.000)	0 (0.001)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	0 (0.001)
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	0 (0.000)	0 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	1 (0.000)	1 (0.000)	1 (0.000)
SIMP-ZRHS	2 (0.000)	2 (0.000)	2 (0.000)
SIMP-GLS	0 (0.000)	0 (0.000)	0 (0.000)
DEMOD-LIN	1 (0.000)	1 (0.000)	1 (0.000)
FERT-TSOS	1 (0.000)	1 (0.000)	1 (0.000)
SATUR-LIN	1 (0.001)	1 (0.000)	1 (0.000)
TRIV-IDEALS	0 (0.000)	0 (0.000)	0 (0.000)
INT-DOM-ZPB	0 (0.000)	0 (0.000)	0 (0.000)
	0.002	0.003	0.005

P23	calc-0	calc-1	calc-2
#(Proof-tree)	8	8	8
QEPCAD	-	-	4 (0.003)
SIMP-GLS	8 (0.000)	4 (0.000)	-
SIMP-ZRHS	8 (0.000)	4 (0.000)	-
SPLIT-INEQS (MAX-SPLITS:=12)	0 (0.000)	0 (0.000)	-
DEMOD-NUM	0 (0.000)	0 (0.000)	-
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	4 (0.000)	4 (0.001)
	0.061	0.003	0.004

Bibliography

[AB98] G. B. Alan and A. Borning. The Cassowary Linear Arithmetic Constraint Solving Algorithm. *ACM Transactions on Computer Human Interaction*, 1998.

[ACM84] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical Algebraic Decomposition. I. The Basic Algorithm. *SIAM Journal on Computing*, 13(4):865–877, November 1984.

[AD99] T. Sturm A. Dolzmann. Redlog User Manual - Edition 2.0. MIP-9905, 1999.

[ADF95] Jürgen Avenhaus, Jörg Denzinger, and Matthias Fuchs. DISCOUNT: A System for Distributed Equational Deduction. In Jieh Hsiang, editor, *Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 397–402. Springer Berlin / Heidelberg, 1995.

[AF06] Jeremy Avigad and Harvey Friedman. Combining Decision Procedures for the Reals. In *Logical Methods in Computer Science*, 2006.

[AP10] Behzad Akbarpour and Lawrence Paulson. MetiTarski: An Automatic Theorem Prover for Real-Valued Special Functions. *Journal of Automated Reasoning*, 44(3):175–205, March 2010.

[BD94] L. Bachmair and N. Dershowitz. Equational Inference, Canonical Proofs, and Proof Orderings. *Journal of the ACM*, 42(2), 1994.

[BG94] Leo Bachmair and Harald Ganzinger. Buchberger's Algorithm: A Constraint-Based Completion Procedure. In Jean-Pierre Jouannaud, editor, *CCL*, volume 845 of *Lecture Notes in Computer Science*, pages 285–301. Springer, 1994.

[BG06] F. Benhamou and L. Granvilliers. *Continuous and Interval Constraints*. Elsevier Science Publishers Ltd., 2006.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Bro01] Christopher W. Brown. Improved Projection for Cylindrical Algebraic Decomposition. *Journal of Symbolic Computation*, 32:447–465, November 2001.

[Bro04] Christopher W. Brown. QEPCAD-B: A System for Computing with Semi-algebraic Sets via Cylindrical Algebraic Decomposition. *SIGSAM Bull.*, 38:23–24, March 2004.

[BS95] Gopalan Balaji and J. Seader. Application of interval Newton's Method to Chemical Engineering Problems. *Reliable Computing*, 1:215–223, 1995. 10.1007/BF02385253.

[BT07] Clark Barrett and Cesare Tinelli. CVC3. In Werner Damm and Holger Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.

[Buc65] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, Austria, 1965.

[Buc79] Bruno Buchberger. A Criterion for Detecting Unnecessary Reductions in the Construction of Groebner Bases. In *EUROSAM '79: Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, pages 3–21, London, UK, 1979. Springer-Verlag.

[Buc84] Bruno Buchberger. A Critical-pair/Completion Algorithm for Finitely Generated Ideals in Rings. In *Proceedings of the Symposium "Rekursive Kombinatorik" on Logic and Machines: Decision Problems and Complexity*, pages 137–161, London, UK, 1984. Springer-Verlag.

[Buc87] Bruno Buchberger. History and Basic Features of the Critical-pair/Completion Procedure. *J. Symb. Comput.*, 3(1-2):3–38, 1987.

[Can93] John Canny. Improved Algorithms for Sign Determination and Existential Quantifier Elimination. *The Computer Journal*, 36:409–418, 1993.

[Car06] Fabrizio Caruso. The SARAG Library: Some Algorithms in Real Algebraic Geometry. In Andrés Iglesias and Nobuki Takayama, editors, *Mathematical Software - ICMS 2006*, volume 4151 of *Lecture Notes in Computer Science*, pages 122–131. Springer Berlin / Heidelberg, 2006.

[Car10] Jacques Carette. Personal Communication, June 2010.

[CG02] M. Ceberio and L. Granvilliers. Horner's Rule for Interval Evaluation Revisited. *Computing*, 69(1):51–81, 2002.

[CH91] G. E. Collins and H. Hong. Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *Journal of Symbolic Computation*, 12(3):299–328, sep 1991.

[CKR04] M. Caboara, M. Kreuzer, and L. Robbiano. Efficiently Computing Minimal Sets of Critical Pairs. *Journal of Symbolic Computation*, 38(4):1169 – 1190, 2004. Symbolic Computation in Algebra and Geometry.

[Cle87] J.G. Cleary. Logical Arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[CLO07] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra (3rd edition)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[CMXY09] Changbo Chen, Marc Moreno Maza, Bican Xia, and Lu Yang. Computing Cylindrical Algebraic Decomposition via Triangular Decomposition. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, pages 95–102, New York, NY, USA, 2009. ACM.

[CRS02] Robert Cori, Dominique Rossin, and Bruno Salvy. Polynomial Ideals for Sandpiles and their Gröbner Bases. *Theor. Comput. Sci.*, 276(1-2):1–15, 2002.

[Dav09] James Davenprt. Personal Communication, 2009.

[DdM06] B. Dutertre and L. de Moura. *The Yices SMT Solver*. SRI International, 2006.

[dMP09] Leonardo de Moura and Grant Olney Passmore. On Locally Minimal Nullstellensatz Proofs. In *SMT '09: Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, pages 35–42, New York, NY, USA, 2009. ACM.

[dMP11] Leonardo de Moura and Grant Olney Passmore. Orchestrating Decision Engines (lecture). Deduction at Scale 2011: Max Planck Society, Ringberg Castle, Tegernsee, Germany, March 2011.

[dMRS04] L. de Moura, H. Rueß, and N. Shankar. Justifying Equality. In *PDPAR'04*, 2004.

[Dol00] Andreas Dolzmann. *Algorithmic Strategies for Applicable Real Quantifier Elimination*. PhD thesis, Universität Passau, Innstrasse 29, 94032 Passau, 2000.

[DSS04] Andreas Dolzmann, Andreas Seidl, and Thomas Sturm. Efficient Projection Orders for CAD. In *ISSAC '04: Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, pages 111–118. ACM, 2004.

[Fau99] Jean Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases (F4). *Journal of Pure and Applied Algebra*, 1999.

[Fau02] Jean Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5). In *ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, New York, NY, USA, 2002. ACM.

[FHR+07] M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *JSAT—Journal on Satisfiability, Boolean Modeling and Computation, Special Issue on SAT/CP Integration*, 1:209–236, 2007.

[GB06] Laurent Granvilliers and Frédéric Benhamou. RealPaver: An Interval Solver using Constraint Satisfaction Techniques. *ACM TRANS. ON MATHEMATICAL SOFTWARE*, 32:138–156, 2006.

[GKW03] Johannes Grabmeier, Erich Kaltofen, and Volker Weispfenning, editors. *Computer Algebra Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 2003.

[GM88] Rüdiger Gebauer and H. Michael Möller. On an Installation of Buchberger's Algorithm. *Journal of Symbolic Computation*, 6(2/3):275–286, 1988.

[Gra95] Peter Graf. Substitution Tree Indexing. In *RTA '95: Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, pages 117–131, London, UK, 1995. Springer-Verlag.

[Gra01] Laurent Granvilliers. On the Combination of Interval Constraint Solvers. *Reliable Computing*, 7:467–483, 2001. 10.1023/A:1014750702474.

[Har07] John Harrison. Verifying Nonlinear Real Formulas via Sums of Squares. In *Proceedings of the 20th international conference on Theorem proving in higher order logics*, TPHOLs'07, pages 102–118, Berlin, Heidelberg, 2007. Springer-Verlag.

[Has07] Brandon Hassett. *Introduction to Algebraic Geometry*. Cambridge University Press, 2007.

[HJVE01] T. Hickey, Q. Ju, and M. H. Van Emden. Interval Arithmetic: From Principles to Implementation. *J. ACM*, 48(5):1038–1068, 2001.

[HMK97] P. Van Hentenryck, D. Mcallester, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM Journal on Numerical Analysis*, 34:797–827, 1997.

[Hon91] Hoon Hong. Comparison of Several Decision Algorithms for the Existential Theory of the Reals. Technical report, RISC (Research Institute for Symbolic Computation), 1991.

[JPP10] Paul B. Jackson, Lawrence C. Paulson, and Grant Olney Passmore. Automatic Proof Procedures for Polynomials and Special Functions. EPSRC grant proposal, April 2010.

[Kal85] Erich Kaltofen. Sparse Hensel Lifting. In Bob Caviness, editor, *EUROCAL '85*, volume 204 of *Lecture Notes in Computer Science*, pages 4–17. Springer Berlin / Heidelberg, 1985.

[Ken39] E. C. Kennedy. Concerning Upper and Lower Bounds of the Roots of a Real Algebraic Equation. *National Mathematics Magazine*, 14(2):pp. 76–80, 1939.

[KMM00] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[KP10] Florent Kirchner and Grant Passmore. Thinking Outside the (Arithmetic) Box: Certifying RAHD Computations. In *Workshop Proceedings for Logics for System Analysis (short paper)*, 2010.

[Krä06] Walter Krämer. Generalized Intervals and the Dependency Problem. *Proc. Appl. Math. Mech.*, 6(1), 2006.

[Kri64] Jean-Louis Krivine. Anneaux Préordonnés. *Journal d'Analyse Mathématique*, 12:307–326, 1964.

[Lap06] Santiago Laplagne. An Algorithm for the Computation of the Radical of an Ideal. In *International Symposium on Symbolic and Algebraic Computation*, 2006.

[LR08] Jinhu Li and Jeffrey Scott Racine. Maxima: An Open Source Computer Algebra System. *Journal of Applied Econometrics*, 23(4):515–523, 2008.

[Mah06] Assia Mahboubi. *Contributions à la Certification des Calculs dans R : Théorie, Preuves, Programmation*. PhD thesis, l'Université de Nice Sophia Antipolis, 2006.

[Mar08] Murray Marshall. *Positive Polynomials and Sums of Squares*, volume 146 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2008.

[MB08] Leonardo De Moura and Nikolaj Björner. Z3: An efficient SMT solver. In *TACAS'08*, 2008.

[McC93] Scott McCallum. Solving Polynomial Strict Inequalities using Cylindrical Algebraic Decomposition. *The Computer Journal*, 36(5), 1993.

[McC03] William McCune. OTTER 3.3 Reference Manual, 2003.

[MH05] Sean McLaughlin and John Harrison. A Proof-Producing Decision Procedure for Real Arithmetic. In Robert Nieuwenhuis, editor, *CADE-20: 20th International Conference on Automated Deduction, proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 295–314, Tallinn, Estonia, 2005. Springer-Verlag.

[MO02] Christian Michaux and Adem Ozturk. Quantifier Elimination Following Muchnik. Universite de Mons-Hainaut Preprint Series (#10), April 2002.

[MY73] Joel Moses and David Y. Y. Yun. The EZ GCD algorithm. In *Proceedings of the ACM annual conference*, ACM '73, pages 159–166, New York, NY, USA, 1973. ACM.

[Nas00] J.C. Nash. The (Dantzig) Simplex Method for Linear Programming. *Computing in Science Engineering*, 2(1):29 –31, 2000.

[Neu90] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge [England] ; New York :, 1990.

[NO07] R. Nieuwenhuis and A. Oliveras. Fast Congruence Closure and Extensions. *Inf. Comput.*, 2005(4), 2007.

[OSRSC99] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[Par03] Pablo A. Parrilo. Semidefinite Programming Relaxations for Semialgebraic Problems. *Math. Program.*, 96(2):293–320, 2003.

[Pau87] Lawrence C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, New York, NY, USA, 1987.

[PdM09a] Grant Olney Passmore and Leonardo de Moura. Superfluous S-polynomials in Strategy-Independent Gröbner Bases. *Symbolic and Numeric Algorithms for Scientific Computing, International Symposium on*, 0:45–53, 2009.

[PdM09b] Grant Olney Passmore and Leonardo de Moura. Universality of Polynomial Positivity and a Variant of Hilbert's 17th Problem. In *Proceedings of Automated Deduction: Decidability, Complexity, Tractibility (work in progress tract)*, 2009.

[PdMJ10] Grant Olney Passmore, Leonardo de Moura, and Paul B. Jackson. Gröbner Basis Construction Algorithms Based on Theorem Proving Saturation Loops. In Nikolaj Bjorner, Robert Niewenhuis, Helmut Veith, and Andrei Voronkov, editors, *Proceedings of Schloss Dagstuhl Seminar on Decision Procedures in Hardware, Software and Bioware*, 2010.

[PJ09] Grant Olney Passmore and Paul B. Jackson. Combined Decision Techniques for the Existential Theory of the Reals. In *Calculemus'09: 16th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning*, 2009.

[PQR09] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real World Verification. In *CADE-22: Proceedings of the 22nd International Conference on Automated Deduction*, pages 485–501, Berlin, Heidelberg, 2009. Springer-Verlag.

[PZ04] Florian Pfender and Günter M. Ziegler. Kissing Numbers, Sphere Packings, and Some Unexpected Proofs. *Notices of the American Mathematical Society*, 51:873–883, 2004.

[Rat06] Stefan Ratschan. Efficient Solving of Quantified Inequality Constraints over the Real Numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.

[Rio03] Renaud Rioboo. Towards Faster Real Algebraic Numbers. *J. Symb. Comput.*, 36:513–533, September 2003.

[Rob49] Abraham Robinson. *The Metamathematics of Algebraic Systems*. PhD thesis, University of London, 1949.

[Rob56] Abraham Robinson. *Complete Theories*. Studies in Logic and the Foundations of Mathematics. North Holland, 1956.

[RS04] H. Rueß and N. Shankar. Solving Linear Arithmetic Constraints. Technical Report SRI-CSL-04-01, SRI International, 2004.

[RV03] Alexandre Riazanov and Andrei Voronkov. Limited Resource Strategy in Resolution Theorem Proving. *Journal of Symbolic Computation*, 36(1-2):101–115, 2003.

[SAG03] Manish Sinha, Luke E. K. Achenie, and Rafiqul Gani. Blanket Wash Solvent Blend Design Using Interval Analysis. *Industrial & Engineering Chemistry Research*, 42(3):516–527, 2003.

[Sch04] Hans Schoutens. Muchnik's Proof of Tarski-Seidenberg. Short note, February 2004.

[SCX03] Li Shuang, Xu Caijun, and Wang Xinzhou. Application of interval Newton Method to Solve Nonlinear Equations and Global Optimization. *Geo-Spatial Information Science*, 6:24–27, 2003. 10.1007/BF02826697.

[Sem86] A. Semenov. Decision Procedures for Logical Theories. *Cybernetics and Computer Technology*, 2:134–146, 1986.

[SRV01] R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov. *Term Indexing*, pages 1853–1964. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 2001.

[Ste73] Gilbert Stengle. A Nullstellensatz and a Positivstellensatz in Semialgebraic Geometry. *Mathematische Annalen*, 207:87–97, 1973. 10.1007/BF01362149.

[Str00] Adam Strzebonski. Solving Systems of Strict Polynomial Inequalities. *Journal of Symbolic Computation*, 29(3):471 – 480, 2000.

[Stu02] Bernd Sturmfels. *Solving Systems of Polynomial Equations*, volume Cbms Regional Conference Series in Mathematics. American Mathematical Society, 2002.

[SV00] A. Shen and N. K. Vereshchaigin. Languages and Calculi. *Moscow Center for Continuous Mathematical Education*, 2000.

[SVJ00] Zsuzsanna Szab, Mikls Vargyas, and A. Peter Johnson. Novel Treatment of Conformational Flexibility Using Interval Analysis. *Journal of Chemical Information and Computer Sciences*, 40(2):339–346, 2000. PMID: 10761137.

[Tar48] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. RAND Corporation, 1948.

[Tiw05a] Ashish Tiwari. An Algebraic Approach for the Unsatisfiability of Nonlinear Constraints. In L. Ong, editor, *Computer Science Logic, 14th Annual Conf., CSL 2005*, volume 3634 of *LNCS*, pages 248–262. Springer, August 2005.

[Tiw05b] Ashish Tiwari. Tiwari's Implementation of a Variant of the Tiwari Positivstellensatz Method. `http://www.csl.sri.com/users/tiwari/html/csl05b.html`, 2005.

[Wei97] Volker Weispfenning. Quantifier Elimination for Real Algebra - the Quadratic Case and Beyond. *Appl. Algebra Eng. Commun. Comput.*, 8(2):85–101, 1997.

[Yun76] David Y.Y. Yun. On Square-free Decomposition Algorithms. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, SYMSAC '76, pages 26–35, New York, NY, USA, 1976. ACM.