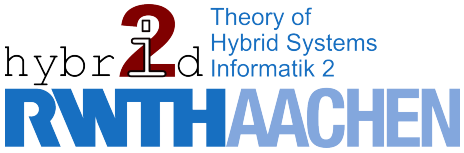


Bachelor of Science Thesis

On Gröbner Bases in SMT-Compliant Decision Procedures

Sebastian Junges

Supervisors:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl
Advisor:
Dipl. Inform. Ulrich Loup

February 26, 2013

Abstract

Modern satisfiability solvers are able to determine satisfiability of a given propositional logic formula very efficiently. Satisfiability modulo theories (SMT) is an approach to use solvers to determine the satisfiability of formulae from the first order logic over some theories.

This thesis aims at the development of methods for deciding consistency of sets of polynomial constraints over the real numbers, which have a decent performance when embedded into an SMT solver. Gröbner bases and the Weak Nullstellensatz allow deciding consistency over the complex numbers. Since Gröbner basis are frequently used and are subject to a lot of active research, the existing algorithms are highly optimised. In this thesis a well-known algorithm is implemented and extended to make the method SMT-compliant.

To decide the unsatisfiability over the real numbers, an application of the Real Nullstellensatz is implemented, in which existing methods for semidefinite programming are combined with Gröbner bases to find sums of squares, which are potential witnesses for unsatisfiability.

The experimental results show some promising applications, which could be further improved by the implementation of the ideas from the thesis' comprehensive overview over both theoretical and technical enhancements.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Sebastian Junges
Aachen, den 22. August 2012

Acknowledgements

There are numerous people who supported me during my work on my bachelor thesis. First of all, I owe my thanks to my supervisor Prof. Dr. Erika Ábrahám. She has supported me for the last couple of years and gave me the great opportunity to be part of her research team. Moreover, she provided very detailed comments on the thesis, far more than I could have expected. I am deeply grateful to my advisor Ulrich Loup, who was available at day- and nighttime to answer all kind of questions and give some valuable feedback. The topic of this thesis is originated in some fruitful discussions we had. I would also like to thank Florian Corzilius for all the work he put into SMT-RAT in order to make it easier for me to develop my SMT-RAT module. The dialogues brought me a deeper understanding of SMT and SMT-RAT. To all in the Theory of Hybrid Systems group, I want to say "Thank You!" for all the help, fun, and meals.

Contents

Chapter 1

Introduction

The *satisfiability problem* (SAT) poses the question whether a propositional logic formula is satisfiable, i.e., whether there is an assignment to the variables such that the formula evaluates to `true`. Many problems from industry can be modelled as a satisfiability problem and therefore, a lot of active research aims at the development of efficient solvers for instances of this problem.

Often, a more expressive modelling language is desired, extending the propositional logic with theories from first-order logic. The combination of theory solvers with SAT solvers is called *Satisfiability Modulo Theories* (SMT) solving. Theory solvers used in SMT should meet some requirements in order to make the interaction with the SAT solver work efficiently.

We focus on a specific theory, the existential fragment of the theory of the real ordered field (*real algebra*). This is an expressive, but still decidable theory [Tar51]. Some well-known decision procedures are the cylindrical algebraic decomposition method and the virtual substitution method. Application areas include the verification of safety-critical programs or models thereof.

From results in automatic theorem proving by [PPdM12] inspired, it seems important to compose a theory solver using several different approaches, as each of the methods may be advantageous for a specific fragment of the theory. Our main focus is on equations. Given a set of polynomials in several variables and of arbitrary degree, we are interested in the common zeroes, i.e., the assignments to the variables such that all polynomials evaluate to 0. More specifically, we are interested in the question if the set of common zeroes is empty over the real numbers.

Over the *complex* numbers, the famous Hilbert's Nullstellensatz gives criteria for emptiness. Evaluation of these criteria can be done using the *Gröbner basis* of a given set of polynomials. The Gröbner basis consists of a set of polynomials, which have the same set of common zeroes as the original polynomials, but it has some properties which simplify checking properties of the set, e.g., whether the set of common zeroes is empty over the complex numbers.

To decide whether the set of common *real* zeroes of a Gröbner basis is empty, one can apply the Real Nullstellensatz by Stengle [Ste74]. However, to apply this theorem efficiently, we have to search for a witness. An approach by Platzer et al. in [PQR09] yields feasible experimental results, which might lead to an efficient SMT procedure.

Contribution

In this thesis we take two heavily used procedures and fit them into an SMT framework in order to gain a speed-up on the detection of conflicts in the set of equalities.

We start by a thorough introduction into the algebraic notions as a theoretical foundation. Based on these we are able to apply results from computational algebra to develop the two new methods which we have adapted, implemented and integrated into the SMT framework.

The first method, based on computing Gröbner bases, determines inconsistencies of equalities over the *complex* numbers. There is a lot of research related to Gröbner basis algorithms, but as far as we know, none of the existing algorithms is SMT-compliant. Another issue is that related research focuses more on algebraically hard problems, which have a structure different than the problems we know from SMT.

We implemented a well-known variant of the *Buchberger algorithm* as well as all dedicated data structures the algorithm uses. The data structures are designed with the structure of typical SMT problems in mind. We extended the algorithm to be SMT-compliant and wrapped it in a Gröbner basis module which contains the SMT-compliant method as well as the interaction with the other modules. We discussed and integrated two methods to factor inequalities into the computation. Moreover, we provide a deeper understanding of the theoretical and technical enhancements that can be made to such calculations.

The second method uses the computed Gröbner basis and is able to determine inconsistency of equalities over the *real* numbers. It is an application of the *Real Nullstellensatz* in which we search for sums of squares as witnesses for unsatisfiability. The approach is based on the ideas from Platzer et al. in [PQR09]. We discuss the original approach and provide a preliminary implementation which uses a numerical library for semidefinite programming. The resulting algorithm is integrated into our Gröbner basis module.

The algebraic components of our module are implemented in GiNaCRA[1]. GiNaCRA is an open-source C++ library for real algebra. The module itself, as well as the major part of our second method, is integrated in SMT-RAT[2]. SMT-RAT is a modular C++ framework for the development of real algebra solvers within SMT. Our implementations will be part of the next release, and are available upon request[3].

Our implemented methods significantly reduce the computation time on several input instances, as we may conclude based on numerous benchmarks, whose results are shown in the section on experimental results.

Related work

Several open computer algebra systems include Gröbner basis algorithms, e.g., Macaulay [GS], Reduce/Redlog [DS97a] and Singular [DGPS12]. Some of the algorithms used in these systems are described in [Bri10], [GGV10], [EP11]. An extensive treatment of technical improvements for Gröbner bases computations is given in [RS]. Using Gröbner bases as a simplification step in quantifier

[1] Available from `http://ginacra.sourceforge.net/`
[2] Available from `http://smtrat.sourceforge.net/`
[3] mail to `sebastian.junges@rwth-aachen.de`.

elimination is mentioned in [DS97b]. All these contributions are superior in terms of both theoretical as technical optimisation, but none regards the needs of an SMT-compliant algorithm and the different structure of problems handled in SMT.

The Real Nullstellensatz is successfully applied in [PQR09]. Applications of the more general Positivstellensatz are given in [Par03], [Tiw05], and [Har07]. Although these approaches handle inequations directly, they require more effort to be applied because three different witnesses have to be found. Furthermore, these approaches do not require a Gröbner basis as input for the computation.

Besides SMT-RAT, we are only aware of one other SMT solver which can handle highly non-linear instances, that is Z3 [dMB08]. Z3 has implemented saturation algorithms for Gröbner bases, which aims at large, largely linear input sets [PdMJ10], but the work on SMT-compliance in [dMP09] is preliminary and has a more theoretical nature. There seems to be no efficient integration of Gröbner bases computations for input sets with higher degrees. Other existing SMT solvers capable of non-linear real algebra are iSAT [FHT$^+$07] and CVC3 [BT07], but these are not complete and are more focused on the mainly linear fragments.

Structure of the thesis

The remaining part of the thesis is structured as follows: In Chapter 2 we discuss SMT solving and requirements for the embedded theory solvers. We then shortly present the framework SMT-RAT. In Chapter 3 we give the foundations of our theory in algebraic terms and introduce Gröbner bases. In Chapter 4 we then develop an SMT-compliant Gröbner basis module and discuss several enhancements. In Chapter 5 we show a possible way to apply the Real Nullstellensatz. In Chapter 6 some experimental results are given. In Chapter 7 we conclude the thesis and give our view on future work.

Chapter 2

SMT solving and real algebra

In this chapter, we describe how the *Satisfiability Modulo Theories* (SMT) solving approach works, and give some desirable features for the SMT components we develop. Finally, we describe the SMT-RAT framework in which we embedded our methods.

2.1 SAT solving

Satisfiability (SAT) *checking* is a well-known problem in computer science. Given a propositional logic formula φ, the SAT problem is to decide whether there exists an assignment of truth values to the variables of φ such that φ evaluates to `true`. If such an assignment exists, we call a formula *satisfiable* (`sat`), otherwise *unsatisfiable* (`unsat`).

Example 2.1.1. *We give two propositional logic formulae:*

- $\varphi = (\neg x \vee y) \wedge (x \vee \neg y \vee z)$ *is* `sat`. *($x =$* `false`*, $y =$* `false`*, $z =$* `true`*).*
- $\psi = (\neg x \vee y) \wedge (x \vee y) \wedge \neg y$ *is* `unsat`.

Although SAT checking is computationally hard, modern solvers are capable to solve very large problem instances efficiently. We regard the family of DPLL SAT solvers. These solvers increase a partial assignment until it is either a full satisfying assignment or they find a conflict. A DPLL solver makes decisions about the value of a variable and then propagates the implications of this decision. Then, if no conflict is detected, a new decision is made. If all variables already have an assignment, the problem is satisfiable. If a conflict is found, the solvers undo those decisions which caused this conflict and 'learn' a new clause describing the conflict's reason. If no new decisions can be made, the problem is unsatisfiable.

Note that DPLL solvers need their input formula to be in *conjunctive normal form* (CNF). However, any Boolean formula can be transformed into an equisatisfiable CNF in linear time and space by Tseitin's encoding [Tse83].

A detailed description of DPLL solvers and their enhancements can be found in e.g. [KS08] and [BHvMW09]. The actual implementation we use is based on the SAT solver MiniSat [ES04].

2.2 SMT solving

Often, propositional logic does not suffice to describe a problem. One would like to have a more expressive logic, for instance some fragment of the first-order logic over some theory. In this fragment, the interpretation of symbols should be fixed. For example, when someone is interested in equality of integers, the symbol '=' should have the standard interpretation. Adding axioms is not always possible[1] and the performance of this solution makes these solvers inapplicable. A better solution is to use dedicated methods tailored to the specific theory.

However, when developing dedicated methods, it is an asset if we can use the highly evolved SAT solvers for the Boolean structure. This approach is called *Satisfiability Modulo Theories* (SMT) solving. We can distinguish between two types of SMT solving. The *eager* approach converts input formulae into equisatisfiable propositional formulae. The *lazy* approach uses an inference system specialised on the theory. We use the last one, as it gives us the possibility to use results from the theory, in our case the real arithmetic.

2.2.1 Lazy SMT solving

The method we use for lazy SMT solving is based on the DPLL(T) framework, which is a combination of a DPLL-based SAT solver with a dedicated theory solver. Our short introduction follows [ÁCLS10]. We start with explaining the basic scheme as depicted in Figure 2.1. Definitions are postponed to Section 2.3, where we define the chief terms with respect to a concrete theory.

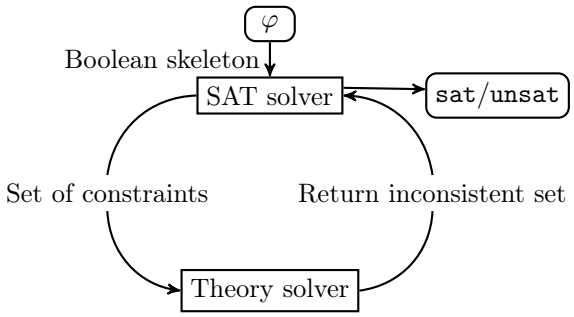


Figure 2.1: Scheme for lazy SMT solving.

For a given Boolean combination φ of constraints, the SAT solver makes an abstraction replacing all the theory constraints by fresh Boolean variables. This yields the Boolean skeleton of φ. The SAT solver then looks for a satisfying solution, and if it does not succeed, immediately returns `unsat`. Otherwise, the theory solver gets the constraints matching the satisfying assignment and it is called for a consistency check. The theory solver checks if the received set of constraints is *consistent* within the theory. If so, the theory solver returns to the SAT solver, which returns `sat`. If the set of constraints is not consistent within the theory, the theory solver returns a set of inconsistent constraints to the SAT solver, which adds the corresponding Boolean variables in a conflict clause and proceeds then, as if it would have found a Boolean conflict.

[1] For the theory of the reals, we would need an infinite set of formulae from the first-order theory.

This scheme is enhanced in *less lazy solving* such that *theory calls* can also happen for partial (Boolean) assignments. In this case, whenever the theory solver returns `sat`, the SAT solver has to extend the assignment, if the assignment is partial, or return `sat`, if the assignment is already a full assignment.

Example 2.2.1. *Given the input formula*

$$\varphi = (x - 1 = 0) \wedge (xy = 0) \wedge ((x + y = 0) \vee (2y = 0)),$$

the SAT solver first generates the Boolean skeleton and a mapping:

$$\varphi_B = a \wedge b \wedge (c \vee d)$$

$$\begin{aligned} a &\mapsto (x - 1 = 0) \\ b &\mapsto (xy = 0) \\ c &\mapsto (x + y = 0) \\ d &\mapsto (2y = 0). \end{aligned}$$

Let us assume that the SAT solver directly assigns a and b to `true`*, and then decides to make a theory call. The theory solver would return* `true` *as* $\{x - 1 = 0,\ xy = 0\}$ *is consistent. The call was on a partial assignment, so the SAT solver extends the assignment. Assume that the SAT solver decides to add c to the assignment. The theory solver receives the new constraint, and now returns the set as conflict. The SAT solver extends the Boolean skeleton with the learned clause:*

$$\varphi_B = a \wedge b \wedge (c \vee d) \wedge (\neg a \vee \neg b \vee \neg c),$$

leading to the assignment $a = b = d =$ `true`, $c =$ `false`*. As constraint set*[1] *we get* $\{x - 1 = 0, xy = 0, 2y = 0\}$*, which is consistent. Since the assignment yields* `true` *in the Boolean skeleton, the SAT solver can return* `sat`*.*

In order to make this scheme efficient, it is important to provide an efficient interaction between the SAT solver and the theory solver. Three requirements for efficient interaction are:

- *Incrementality*: The theory solver should use the result from a previous check whenever the input constraint set is extended.
- *Backtrackability*: The theory solver should be able to restore a previous state whenever the input set is decreased.
- *Small conflict-set generation*: The theory solver should be able to return inconsistent sets of constraints which are as small as possible.

Definition 2.2.1 (SMT-compliant)**.** *We call a theory solver* SMT-compliant *if it supports incrementality, bactrackability and the generation of small conflict sets.*

Notice that there are a couple of other properties which make a theory solver work even more efficient. These are not in the scope of our definition. We mention two additional features which further improve the interaction.

[1] The input formula is in negation normal form (NNF), i.e., negation only occurs in front of Boolean variables. If the input is in NNF, then only constraints whose corresponding variable is assigned to `true` must be passed on to the theory solver.

- *Informing*: The theory solver should have knowledge about constraints which might be added in future theory calls.
- *Theory deduction*: The theory solver should have the ability to tell the SAT solver a pair of a set of constraints C and a constraint c, such that if all constraints in C evaluate to `true`, then c evaluates to `true`.

2.3 Theory of the reals

In this thesis, we consider decision procedures for *real arithmetic* (RA). An RA formula φ is an arbitrary Boolean combination of constraints c. Constraints compare multivariate polynomials p over a set V of variables with zero. Multivariate polynomials are variables and constants, arbitrarily combined by addition, subtraction and multiplication. We define them by a grammar with $x \in V$:

$$\begin{array}{lcl} p & ::= & 1 \mid x \mid (p+p) \mid (p-p) \mid (p \cdot p) \\ c & ::= & p = 0 \mid p < 0 \mid p > 0 \\ \varphi & ::= & c \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\exists x \varphi) \end{array}$$

The semantics for such formulae is as usual. Several Boolean operators can be added as syntactic sugar, moreover, we can also define the relations $\leq, \geq, \neq$ in constraints.

A special case of RA is *non-linear real arithmetic* (NRA). In NRA, there exist constraints which are not linear, i.e., whose degree is larger than one (see Definition 3.1.5). If, in contrast, all constraints are linear, i.e., the degree of all constraints is less than two, we call the fragment of RA *linear real arithmetic* (LRA). For LRA highly efficient methods are available. In this thesis, we focus on constraints which are non-linear.

Next we fix some notions. A more formal treatment of the theory is given in Chapter 3.

Definition 2.3.1 (Variable assignment)**.** *A (real valued)* variable assignment $\alpha : V \to \mathbb{R}$ *is a map from the set of variables V to the value domain $\mathbb{R}$.*

Definition 2.3.2 (Satisfying assignment)**.** *A set C of constraints is* consistent *if it has a* satisfying assignment*, i.e., if there exists a variable assignment such that all constraints $c \in C$ evaluate to* `true`*. A set C of constraints is a* conflict set*, also called* inconsistent*, if there exists no satisfying assignment for C.*

Definition 2.3.3 (Theory call)**.** *A* theory call *returns for a given set C of constraints either a conflict set $C' \subset C$ or the empty set in case C is consistent.*

2.3.1 Common approaches

We hereafter shortly introduce the two major procedures which are mostly used for non-linear problems in the context of SMT solving.

Virtual substitution Virtual substitution (VS) is a method originally developed by Weispfenning in [Wei93]. It is a quantifier elimination procedure based on a finite abstraction of the state space, substituting variables by a finite

set of test candidates. Instead of a real substitution, for an input formula φ it produces a set of formulae Ψ such that φ is unsatisfiable if and only if all $\psi \in \Psi$ are unsatisfiable. Since this substitution only works for polynomials of bounded degree, it is incomplete, but it turned out to be very efficient on many problems. An SMT-compliant version is given in [CA11].

Cylindrical algebraic decomposition Cylindrical algebraic decomposition (CAD) is a method originally developed by Collins in [Col75]. Instead of transforming the input formula, the CAD partitions the solution space into regions over which all input polynomials are sign invariant. This is done in two phases: projection and lifting. The projection is done by successive elimination of variables. When there is only one variable left, the univariate solutions can be easily calculated. In the lifting phase, we retrieve sample points for all cells from the partitioned solution space. Since the representation of points in the solution space is far from trivial, the CAD generally performs better when it gets (strict) inequalities. Despite this, the CAD is a complete decision procedure.

2.3.2 Applications

Nowadays, more and more digital systems interact with their physical environment. Often, these systems are safety-critical, e.g., control systems in transportation. The behaviour of such systems can be modelled by means of *hybrid automata*. Hybrid automata can be verified to fulfil certain properties. One approach to do this is bounded model checking [Arm03]. During bounded model checking, real arithmetic formulae with a Boolean structure have to be checked for satisfiability. Therefore, there is a great interest in SMT solving within this community (see e.g. [CMT12]).

Other applications for SMT solving are verification of programs involving floats and the verification of formulae involving special functions [AP10].

2.4 SMT-RAT

We integrated our modules in SMT-RAT [CLJÁ12]. SMT-RAT is a modular C++ framework to support the implementation of SMT solvers for non-linear real arithmetic. Different methods and approaches to solve or simplify sets of constraints can be implemented as *modules*. At compile-time, these modules can be composed into a *hierarchy* together with a *strategy* which, during runtime, decides which module should handle the input. In the version 0.3, there are modules for CAD, Virtual Substitution as well as some preprocessing and simplification modules.

The framework was slightly redesigned recently, therefore, we describe the most important parts of the interface here. A more detailed description can be found in the user's manual [CLJÁ].

2.4.1 General overview

The general idea of the framework can be seen in Figure 2.2. The modules, depicted as boxes contain different methods and/or different heuristics for these methods. The links between the modules model the strategy. The approach to

solve a problem instance is that a module might work on a fragment of the input and then pass a modified part of the formula to another module, which might contain a more efficient method to solve this new, equisatisfiable formula. The order of these calls and the specific modules which are called is described by the strategy. We call a module B, which is called after another module A, the *backend module* of A. The interaction between two successive modules is realised

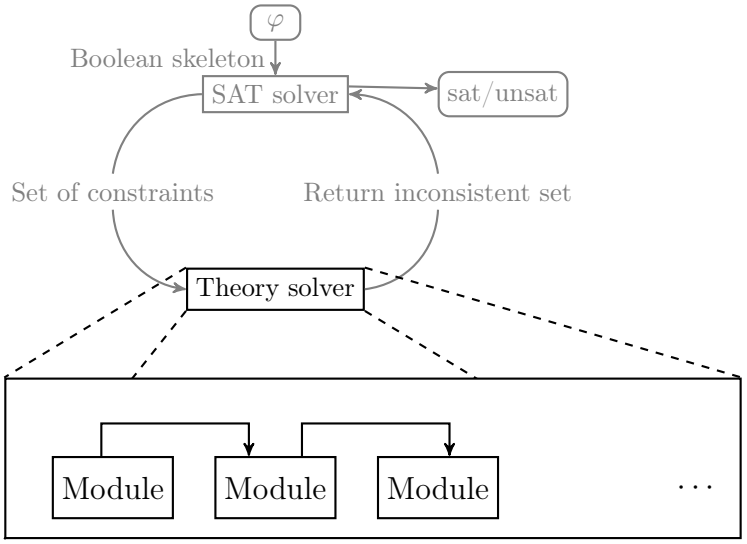


Figure 2.2: Modules and strategy in SMT-RAT.

by sharing a common set of constraints[1]. Every module has a *received formula* and a *passed formula*. The passed formula is the same as the received formula of its direct successors. In Figure 2.3 the Module A has a received formula φ_1 and a passed formula φ_2. Module B has φ_2 as its received formula.

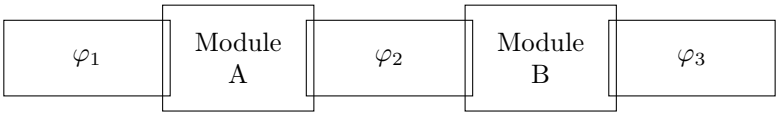


Figure 2.3: Two modules with their in- and output.

Modules should never change their received formula, as there might be other modules also working on this formula. Moreover (see Figure 2.4) whenever a backend module returns a conflict set with respect to the passed formula, it is important to translate this to a conflict set with respect to the received formula such that the SAT solver finally gets a conflict set with respect to its passed constraints. In this manner, the SAT solver is able to understand the conflict and learn its conflict clause. In Figure 2.4 Module C finds a conflict. It returns a subset of its received constraints. Module B and all other predecessors translate this subset back.

[1] In fact, they share a common formula, but for the theory solvers we use, this is always a conjunction of constraints.

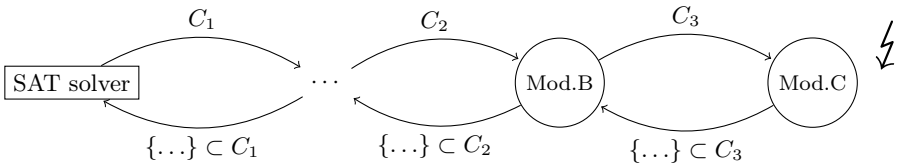


Figure 2.4: Learning a conflict.

Definition 2.4.1 (Reason set)**.** *Given two sets C_1, C_2 of constraints, the* reason set *of a constraint $c \in C_2$ is a set $R \subseteq C_1$ such that $\bigwedge_{r \in R} r \to c$.*

The module has an interface which keeps a mapping storing the reason sets for all constraints in the passed formula. This mapping is used for restoring the conflict sets, but besides that, the mapping is also used to update the backends whenever backtracking occurs, such that the backtracking is applied to the passed formula (and thus the backend modules).

2.4.2 Implementing an own module

To implement an own module in SMT-RAT, one has to inherit the functionality from a generic `Module` class. The own class should overload at least the three core methods, which define the behaviour of the module. In order to keep data structures correct, calls to the method in the superclass should be made.

`assertSubformula( constraint ):` From now on, theory calls to the module should consider the constraint in the received formula.

`isConsistent( ):` This method asks the module for consistency of the asserted constraints in the received formula. Besides `true` and `false`, the check can also return `unknown`. In order to have the ability to add backends for this module, a call to `runBackends` should be added at a suitable place after the calculation.

`removeSubformula( constraint ):` This method is basically the inverse of `assertSubformula`. It informs the module that it should no longer take the constraint into account.

2.4.3 Theory solver and strategy

A theory solver built from SMT-RAT modules can be best regarded as a set of rules which define the strategy. Each formula has a set of properties, which are calculated on-demand. There are two major categories of properties. First, those properties describing the formula, e.g., whether or not all constraints are equalities. Second, those which describe whether another module was already called on the formula. Now to define a strategy, we define a set of rules. Each rule consists of a priority, a function which maps properties of a formula to a Boolean value, and a module. The rules are evaluated according to their priority order, and as soon as the function evaluates to true, the corresponding module is attached as *used backend*, and the constraints which have been added

but not yet asserted in this module will be passed as arguments in the calls to `assertSubformula`.

Chapter 3

Consistency for polynomials

In this chapter, we formalise our problem from the previous chapter in terms of real algebra. We regard polynomials as algebraic objects. This enables us to use well-understood theory about polynomials and we can define what consistency means in algebraic terms. We then move on to Gröbner bases, which are an important tool for simplifying the representation of polynomial sets. In the last part, we consider criteria for consistency and shortly show how to apply some of the results to inequalities.

3.1 Consistency as an algebraic notion

We start with some basic algebraic definitions which can be found in, e.g., [BWK93] and [CLO97].

3.1.1 Rings and fields

For our treatment of polynomials, two algebraic structures are elementary.

Definition 3.1.1 (Commutative Ring with 1)**.** *Given a set R and two binary operations, $+$ and $\cdot$, for which the following holds:*

1. $a + (b + c) = (a + b) + c$ *for all* $a, b, c \in R$ *(associative addition).*
2. $a + b = b + a$ *for all* $a, b \in R$ *(commutative addition).*
3. *There is a* $0 \in R$ *such that* $a + 0 = a$ *for all* $a \in R$ *(existence of a zero).*
4. *For all* $a \in R$ *there exists a* $b \in R$ *such that* $a + b = 0$ *(inverse element).*
5. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ *for all* $a, b, c \in R$ *(associative multiplication).*
6. $a \cdot b = b \cdot a$ *(commutative multiplication).*
7. *There is a* $1 \in R$ *such that* $a \cdot 1 = a$ *for all* $a \in R$ *(existence of a one).*

Then $(R, +, \cdot)$ *is a* commutative ring with 1*. Since we only regard commutative rings with a 1, we will simply refer to them as* rings*.*

If the binary operators are clear from the context, we simply omit them.

Definition 3.1.2 (Field)**.** *Given a set K and two binary operation, $+$ and $\cdot$, for which the following holds:*

1. *$(K, +, \cdot)$ is a ring.*
2. *For all $a \in K, a \neq 0$ there exists a $b \in K$ such that $a \cdot b = 1$ (multiplicative inverse element).*

Then $(K, +, \cdot)$ is a field.

Example 3.1.1. *A well-known example for a ring is the set of integers, $\mathbb{Z}$. Well known examples for fields are the sets $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ of rational numbers, real numbers, and complex numbers respectively.*

3.1.2 Polynomials

We now formally define polynomials as algebraic objects. Let in the following $\bar{x} = \{x_1, \ldots, x_n\}$ with $n \geq 1$, be a set of variables.

Definition 3.1.3 (Monomial)**.** *A* monomial *over $\bar{x}$ is a product $\bar{x}^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ for some* exponent vector *$\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$. The set of monomials over $\bar{x}$ is denoted $M_{\bar{x}}$. The* total degree *of a monomial $\bar{x}^\alpha$, denoted by $\mathsf{tdeg}(\bar{x}^\alpha)$, is defined as $\sum_{i=1}^{n} \alpha_i$. If the total degree of a monomial equals zero, it is the* constant monomial 1*. We say that a variable $x_i \in \bar{x}$* occurs *in a monomial $\bar{x}^\alpha \in M_{\bar{x}}$ if $\alpha_i > 0$ and use $x_i \in \bar{x}^\alpha$ as notation.*

Definition 3.1.4 (Operations on monomials)**.** *For $m_1 = \bar{x}^\alpha$ and $m_2 = \bar{x}^\beta$ we define:*

1. multiplication*: $m_1 \cdot m_2 = \bar{x}^\gamma$ with $\gamma_i = \alpha_i + \beta_i$ for all $1 \leq i \leq |\bar{x}|$.*
2. divides*: $m_1 \mid m_2$ if and only if $\alpha_i \leq \beta_i$ for all $1 \leq i \leq |\bar{x}|$.*
3. division*: If $m_1 \mid m_2$ then $\frac{m_2}{m_1} = \bar{x}^\gamma$ with $\gamma_i = \beta_i - \alpha_i$ for all $1 \leq i \leq |\bar{x}|$.*

Polynomials are obtained by addition of monomials multiplied by some coefficients.

Definition 3.1.5 (Polynomial)**.** *Let K be a field. A* polynomial *f over $\bar{x}$, $f = a_1 m_1 + \ldots + a_l m_l$, is a finite sum of pairwise different monomials $m_i \in M_{\bar{x}}$ with coefficents, $a_i \in K$ for all $1 \leq i \leq l$. The* total degree*, $\mathsf{tdeg}(f)$, of a polynomial f is $\max_{i=1}^{l}\{\mathsf{tdeg}(m_i) : a_i \neq 0\}$. If $|\bar{x}| = 1$ we call the polynomial* univariate*, otherwise it is called* multivariate*. Polynomials with $\mathsf{tdeg}(f) = 0$ are called* constant*. Polynomials with $\mathsf{tdeg}(f) = 1$ are called* linear*.*

Example 3.1.2. *The polynomial $f = 2x_1x_2 + x_1$ is a multivariate polynomial with a total degree of 2. The polynomial $g = 3x_1^3 + 2x_1$ is a univariate polynomial with a total degree of 3.*

Definition 3.1.6 (Polynomial ring)**.** *[BWK93, Chapter 2] The set of all polynomials over a given set of variables $\bar{x}$ with coefficients in K is denoted as $K[\bar{x}]$. Together with the addition and multiplication of polynomials, $K[\bar{x}]$ is a ring, which we call the* polynomial ring *over K.*

Polynomials in the context of SMT solving are regarded as functions. The following defines this notion formally.

Definition 3.1.7 (Polynomial evaluation function)**.** *Given a polynomial ring $K[\bar{x}]$, a field N such that $K \subseteq N$, and a polynomial $f = \sum_{j=1}^{m} a_j \bar{x}^{e_j} \in K[\bar{x}]$. Then we can define the following map: $f : N^n \to N$, $f(c_1,\ldots,c_n) = \sum_{j=1}^{m} a_j c_1^{e_{j_1}} \cdots c_n^{e_{j_n}}$. We call $f(c_1,\ldots,c_n)$ the* evaluation *of f under c.*

It follows immediately from the definition that the evaluation can also be applied to single monomials and that the evaluation of a polynomial equals the weighted sum of the evaluations of its monomials.

Definition 3.1.8 (Zero of f)**.** *Let $K \subseteq N$ be fields, and $c_1,\ldots,c_n \in N$. For any polynomial $f \in K[x_1,\ldots,x_n]$, the tuple $(c_1,\ldots,c_n)$ is called a* zero *of f if $f(c_1,\ldots,c_n) = 0$.*

Example 3.1.3. *The function $f = 2x_1x_2 + x_1$ has an evaluation under* (3,2) *which equals $f(3,2) = 2 \cdot 3 \cdot 2 + 3 = 15$.*

3.1.3 Ideals and varieties

In the previous chapter, we discussed that we want to check the consistency of constraints. We now define consistency for polynomials. Notice that we only regard equalities for now.

Definition 3.1.9 (Affine variety)**.** *Let $K \subseteq N$ be fields, and let $f_1,\ldots,f_s$ be polynomials in $K[x_1,\ldots,x_n]$. The set of the* common zeroes *of $f_1,\ldots,f_s$, defined as*

$$V_N(f_1,\ldots,f_s) = \{(c_1,\ldots,c_n) \in N^n : f_i(c_1,\ldots,c_n) = 0 \text{ for all } 1 \leq i \leq s\}$$

is called the affine variety *of $f_1,\ldots,f_s$ over N. We sometimes omit the N if it is clear from the context.*

Consistency of polynomial equations thus reduces to checking if the affine variety of some given polynomials is empty. In order to efficiently work with these varieties, we first make an observation. Given a system of polynomials $f_1,\ldots,f_s \in K[\bar{x}]$ such that for an arbitrary $\bar{c} \in N^n$ it holds that

$$\begin{aligned} f_1(\bar{c}) &= 0 \\ f_2(\bar{c}) &= 0 \\ &\vdots \\ f_s(\bar{c}) &= 0 \end{aligned}$$

then, for this $\bar{c}$ it also holds that for arbitrary $h_1,\ldots,h_s \in K[\bar{x}]$ the linear combination $(\sum_{i=1}^{s} h_i f_i)(\bar{c}) = 0$.

We now introduce ideals as an algebraic object which turns out to be suitable for describing common zeroes.

Definition 3.1.10 (Ideal)**.** *Let R be a ring and $I \subseteq R$. I is an* ideal of R *if*

1. *$0 \in I$.*
2. *$a + b \in I$ for all $a,b \in I$.*
3. *$r \cdot a \in I$ for all $a \in I$ and $r \in R$.*

Lemma 3.1.1 (Generated ideal)**.** *[CLO97, p.31] If $f_1, \ldots, f_s \in K[\bar{x}]$. Then the set*

$$\langle f_1, \ldots, f_s \rangle = \{\sum_{i=1}^{s} h_i f_i : h_i \in K[\bar{x}] \text{ for all } 1 \leq i \leq s\}$$

is an ideal of $K[\bar{x}]$ and we call it the ideal generated by *$f_1, \ldots, f_s$. The polynomials $f_1, \ldots, f_s$ are called the* generators *of $\langle f_1, \ldots, f_s \rangle$.*

Theorem 3.1.2 (Hilbert basis theorem)**.** *[CLO97, p. 74] Every ideal I of $K[\bar{x}]$ is generated by a finite number of polynomials, i.e, $I = \langle f_1, \ldots, f_s \rangle$ for some $f_1, \ldots, f_s \in K[\bar{x}]$.*

As a next step, we consider the correspondence between affine varieties and ideals.

Proposition 3.1.3. *If $\langle f_1, \ldots, f_s \rangle = I \subseteq K[\bar{x}]$ and $\langle g_1, \ldots, g_t \rangle = J \subseteq K[\bar{x}]$ with $I = J$, then $V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$.*

An idea for the proof is based on the observation that every generator of J is a linear combination of generators of I. Therefore for every g_i, $g_i(\bar{c}) = 0$ for all $c \in V(f_1, \ldots, f_s)$. Thus, $V(I)$ is well-defined. Moreover, from the definition we directly get:

Corollary 3.1.4. *If for a given ideal I it holds that $1 \in I$ then $V(I) = \emptyset$.*

Definition 3.1.11. *Let $V \subseteq N^n$ be an affine variety. We define $I(V) = \{f \in K[\bar{x}] : f(\bar{c}) = 0 \text{ for all } \bar{c} \in V\}$.*

The set $I(V)$ is an ideal. However, the ideal-variety correspondence is not as close as we might have hoped for.

Proposition 3.1.5. *If $f_1, \ldots, f_s \in K[\bar{x}]$. Then it holds that $\langle f_1, \ldots, f_s \rangle \subseteq I(V(f_1, \ldots, f_s))$. However, we do not have equality in the general case.*

The proof can be found in [CLO97, p. 33]. A counterexample for the converse is given here.

Example 3.1.4. *At first we notice that $x \notin \langle x^2, y^2 \rangle$, since x has a total degree of one, and all polynomials in $\langle x^2, y^2 \rangle$ are of the form $h_1 x^2 + h_2 y^2$ and thus are either zero or have a total degree of at least 2. It follows directly that $\langle x, y \rangle$ is strictly larger than $\langle x^2, y^2 \rangle$. Now we regard $I(V(x^2, y^2))$. Since $x^2 = y^2 = 0$ implies $x = y = 0$, we get that $V(x^2, y^2) = \{(0,0)\}$. The only thing which is still to be shown is $I(\{(0,0)\}) = \langle x, y \rangle$. Every polynomial of the form $h_1 \cdot x + h_2 \cdot y$ trivially vanishes at (0,0). Moreover, if a polynomial vanishes at (0,0), the constant monomial should have a zero coefficient (and thus be zero, and be in the ideal). All other monomials are multiples of at least x or y, and therefore are in the ideal.*

Before proceeding, we notice that, even in the complex case, it does not hold that $V(I) = V(J) \implies I = J$.

Example 3.1.5. $V(\langle x \rangle) = \{0\} = V(\langle x^2 \rangle)$.

We conclude this section with another positive result on the correspondence between varieties and ideals.

Proposition 3.1.6. *[CLO97, p.32] Let V, W be affine varieties in $K[\bar{x}]$. Then $V \subset W$ if and only if $I(V) \supset I(W)$.*

Intuitively, this proposition states that adding a polynomial constraint may reduce, but never increases the set of common zeroes.

3.2 Gröbner bases

The last section gives rise to two problems.

Question 1 (Reduced basis problem)**.** *How can we simplify the generators of a given ideal?*

This yields a more compact representation of the ideal and thereby a more compact representation for the variety.

Question 2 (Membership problem)**.** *How can we decide whether a polynomial is in a given ideal I?*

Checking whether $1 \in I$ would allow us to check whether $V(I)$ is empty (Lemma 3.1.4). In later chapters, we also want to check other polynomials for membership.

For linear polynomials we can use Gaussian elimination in order to answer both questions. Gaussian elimination is suitable for the linear case due to the fact that different monomials cannot divide each other, which however is possible in the non-linear case.

For univariate polynomials we would use the *greatest common divisor* (gcd) to obtain a single generator for the given ideal, whereas in the multivariate case, it is in general impossible to find such a single generator [BWK93, p. 86]. Membership checking in the univariate case is possible with *polynomial long division* (Example 3.2.2 or [CLO97, Section 1.5]) and then checking whether the remainder is zero.

For the general case, we thus have to generalise both procedures. An extended treatment of the univariate and linear case and a comparison to the general case can be found in [AL94].

3.2.1 Polynomial reduction

In both Gaussian elimination and the Euclidean computation of the gcd, a reduction of polynomials is the core of the algorithm. An important, although seldomly stressed property of these reductions is the order of the monomials, which is important for both correctness and termination. Therefore, we define such monomial orders for the general case.

Definition 3.2.1 (Variable order)**.** *A* variable order *is a linear order on the set of variables $\bar{x}$.*

Linear orders are orders which are *transitive*, *antisymmetric*, and *total*. We will not explicitly describe or refer to the variable orderings. As a convention we order according to the indices (e.g. $x_1 > x_2$) or according to the lexicographic order ($x > y$).

Definition 3.2.2 (Monomial order)**.** *For any $\bar{x}$ and a given variable order $<_{\bar{x}}$ on $\bar{x}$, a linear order $\prec$ on $M_{\bar{x}}$ is a* monomial order *if it satisfies*

1. $1 \prec m \quad \forall m \in M_{\bar{x}} \setminus \{1\}$
2. $m_1 \prec m_2 \implies mm_1 \prec mm_2 \quad \forall m, m_1, m_2 \in M_{\bar{x}}$.

There are several possible monomial orders, we define the four which are most commonly used in both literature and implementations.

Definition 3.2.3. *Let $m_1 = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ and $m_2 = x_1^{\beta_1} \cdots x_n^{\beta_n}$.*

- lexicographic *order:* $m_1 \geq_{lex} m_2 \iff$

$$\exists k\ 1 \leq k \leq n \wedge \alpha_k > \beta_k \wedge \forall i\ 1 \leq i < k \implies \alpha_i = \beta_i$$

- graded lexicographic *order:* $m_1 \geq_{grlex} m_2 \iff$

$$\mathsf{tdeg}(m_1) > \mathsf{tdeg}(m_2) \vee (\mathsf{tdeg}(m_1) = \mathsf{tdeg}(m_2) \wedge m_1 \geq_{lex} m_2)$$

- graded reverse lexicographic *order:* $m_1 \geq_{grrevlex} m_2 \iff$

$$\begin{gathered} \mathsf{tdeg}(m_1) > \mathsf{tdeg}(m_2) \vee \\ (\mathsf{tdeg}(m_1) = \mathsf{tdeg}(m_2) \wedge \\ \exists k\ 1 \leq k \leq n \wedge \alpha_k < \beta_k \wedge \forall i\ k < i \leq n \implies \alpha_i = \beta_i) \end{gathered}$$

The graded orders are called degree-based *orders.*

Example 3.2.1.

$$\begin{array}{lll} x^2y^2 \geq_{lex} x^1y^3 & x^2y^2 \geq_{grlex} x^1y^3 & x^2y^2 \geq_{grrevlex} x^1y^3 \\ x^3y^1 \geq_{lex} x^1y^4 & x^3y^1 \not\geq_{grlex} x^1y^4 & x^3y^1 \not\geq_{grrevlex} x^1y^4 \\ x^2z \geq_{lex} xz^2 & x^2z \geq_{grlex} xz^2 & x^2z \not\geq_{grrevlex} xz^2 \end{array}$$

In general, lexicographic orders tend to yield a basis with some 'nice' properties, however, calculating them is more expensive. In the remainder we will not explicitly refer to the monomial order, and we will use the graded lexicographic order in examples. The next definition makes it easier to reason about polynomials when discussing reduction.

Definition 3.2.4 (Leading monomial and leading coefficient)**.** *Assume a monomial order $<$ and a polynomial $f = \sum_{i=1}^{m} c_i m_i \in K[\bar{x}]$ with $c_i \neq 0$ and $m_j < m_i$ for all $1 \leq i < j \leq m$.*

1. *The* leading monomial *of f, $\mathsf{lm}(f)$, equals m_1.*
2. *The* leading coefficient *of f, $\mathsf{lc}(f)$, equals c_1.*
3. *The* leading monomial with coefficient *of f, $\mathsf{lmc}(f)$, equals $\mathsf{lc}(f) \cdot \mathsf{lm}(f)$.*

Lemma 3.2.1. *[BWK93, p.195] If f,$g \in K[\bar{x}]$ with $f \neq 0$, $g \neq 0$ and f, g both ordered with the same order $<$. Then the following hold:*

1. $\mathsf{lm}(f \cdot g) = \mathsf{lm}(f) \cdot \mathsf{lm}(g)$

2. $\mathsf{lc}(f \cdot g) = \mathsf{lc}(f) \cdot \mathsf{lc}(g)$
3. $\mathsf{lm}(f + g) \leq \max\{\mathsf{lm}(f), \mathsf{lm}(g)\}$.

Notice that we do not have equality, as the leading terms may cancel each other.

We will now generalise the polynomial long division[1] from the univariate case to the multivariate case. We start by giving a little example for the univariate case.

Example 3.2.2. *Dividing $f = 4x^3 + x^2 + 2$ by $g = 2x + 1$ we start with $r = f$ and then we get*

1. $\mathsf{lmc}(r) = 4x^3$ *is dividable by* $\mathsf{lmc}(g) = 2x$ *with factor* $2x^2$. *After the first step we have* $r = 4x^3 + x^2 + 2 - 2x^2(2x + 1) = -x^2 + 2$.
2. $\mathsf{lmc}(r) = -x^2$ *is dividable by* $\mathsf{lmc}(g) = 2x$ *with factor* $-\frac{1}{2}x$. *The remainder now is* $r = -x^2 + 2 - (-\frac{1}{2}x)(2x + 1) = \frac{1}{2}x + 2$
3. $\mathsf{lmc}(r) = \frac{1}{2}x$ *is dividable by* $\mathsf{lmc}(g) = 2x$ *with factor* $\frac{1}{4}$. *The remainder now is* $r = \frac{1}{2}x + 2 - \frac{1}{4}(2x + 1) = \frac{7}{4}$
4. $\mathsf{lmc}(r) = \frac{7}{4}$ *is not dividable by* $\mathsf{lmc}(g) = 2x$.

So as result we get $f = (2x^2 - \frac{1}{2}x + \frac{1}{4})g + \frac{7}{4}$.

We are mostly interested in the remainder. We therefore call such a procedure *reduction*. If the result of a reduction of f is f', we say that f *reduces* to f'. For the multivariate case, we start considering a single reduction step.

Example 3.2.3. *We want to reduce $f = x^2y + y^2$ by $g = xy - 1$, w.r.t. the lexicographic order and $x > y$. The reduction step is possible, since $\mathsf{lmc}(g)$ divides $\mathsf{lmc}(f)$. Thus f reduces to $f' = x^2y + y^2 - x \cdot (xy - 1) = y^2 + x$.*

Definition 3.2.5 (Top reduction)**.** *Let f, f', g be polynomials in $K[\bar{x}]$ with $f \neq 0$, $g \neq 0$. Then f* top-reduces *to f' modulo g if $\mathsf{lm}(g) \mid \mathsf{lm}(f)$ and*

$$f' = f - \frac{\mathsf{lmc}(f)}{\mathsf{lmc}(g)} \cdot g$$

We write $f \xrightarrow{g}_t f'$ for such a step.

In contrast to the univariate case, it is possible that although the leading monomial is not dividable, another monomial in the polynomial is.

Example 3.2.4. *We want to reduce $f = x^2 + xy$ by $g = xy - 1$ w.r.t. the lexicographic order and $x > y$. Top-reduction is not possible, since $\mathsf{lmc}(g)$ does not divide $\mathsf{lmc}(f)$. But $\mathsf{lmc}(g)$ divides xy. Thus $x^2 + xy$ reduces to $x^2 + xy - (xy - 1) = x^2 + 1$.*

Definition 3.2.6 (Reduction)**.** *Let f, f', g be polynomials in $K[\bar{x}]$ with $f \neq 0$, $g \neq 0$ and $f = a_1m_1 + \ldots + a_lm_l$, with $a_i \in K, m_i \in M_{\bar{x}}$ for all $1 \leq i \leq l$. Then* f reduces to f' modulo g *if there exists an i, $0 \leq j \leq l$ s.t. $\mathsf{lm}(g) \mid m_i$ and*

$$f' = f - \frac{a_im_i}{\mathsf{lmc}(g)} \cdot g.$$

We write $f \xrightarrow{g} f'$ for such a step. The polynomial g is called the reductor.

[1] The polynomial long division in its basic form is explained in [CLO97, p. 38].

Since we cannot combine the several generators of an ideal into a single one in the multivariate case [BWK93, p. 86], we have to extend the notion of reduction such that it is able to cope with a set of reductors F.

Definition 3.2.7. *Let f, f' be polynomials in $K[\bar{x}]$ with $f \neq 0$ and $F \subseteq K[\bar{x}]$. Then f* reduces to *f'* modulo *F , written $f \xrightarrow{F} f'$, if there exists a $g \in F$ s.t. $f \xrightarrow{g} f'$. We write $f \xrightarrow{F}{}^{*} f'$ for multiple reduction steps.*

In Algorithm 1 we show the pseudo code for the reduction modulo a set of polynomials, which closely resembles [CLO97, p. 62]. Note that the existence

```
1  Input: f, G = g_1,...,g_s, ≥
2  Output: r s.t. f ->^G* r w.r.t. ≥
3  r = 0; p = f;
4  while (p ≠ 0) {
5    if (∃ i  lmc(g_i) | lmc(p)) {
6      p = p - lmc(p)/lmc(g_i) * g_i;
7    } else {
8      r = r + lmc(p);
9      p = p - lmc(p);
10   }
11 }
```

Algorithm 1: Polynomial reduction.

of a reductor can be easily checked with a while loop, but for an efficient implementation other strategies are better, and the correctness does not depend on the way we check for and select our reductor. The proof for correctness is similar to [CLO97, p. 62]. We only sketch the main ingredients here. The following lemma shows that the term 'reduction' is suitable and is the main ingredient for showing termination.

Lemma 3.2.2. *[BWK93, p. 198] Given f, f', $g \in K[\bar{x}]$. If $f \xrightarrow{g}_t f'$ then $\mathsf{lm}(f) > \mathsf{lm}(f')$.*

The correctness follows from the fact that we apply the definition of reduction whenever we can find a suitable g_i, and, with the lemma above, it can be shown that monomials in r are added in a strict descending monomial order.

Definition 3.2.8 (Normal form)**.** *Given f,$f' \in K[\bar{x}]$ with $f \neq 0$ and $F \subset K[\bar{x}]$. Then $f' = \bar{f}^F$ is the* normal form modulo *F if $f \xrightarrow{F}{}^{*} f'$ and f' cannot be reduced modulo F.*

Proposition 3.2.3. *Let $f, f' \in K[\bar{x}]$ with $f \neq 0$ and $F = \{f_1, \ldots, f_s\} \subset K[\bar{x}]$. If $f \xrightarrow{F}{}^{*} f'$, then there exist $h_1, \ldots, h_s \in K[\bar{x}]$ such that $f = f_1h_1 + \ldots + f_sh_s + f'$.*

This follows directly from the definition, which can be constructively shown by the division variant of Algorithm 1 in [CLO97, Theorem 3].

3.2.2 Ideal reduction

If we want to know whether $f \in \langle f_1, \dots f_s \rangle = I$, we can use the reduction algorithm. If the remainder equals zero, then $f \in I$. However, for the other direction we have a counterexample.

Example 3.2.5. *Given $F = \langle x+1, x \rangle$ and $f = 2x+2$. $f \xrightarrow{x+1} 0$, so there exist $h_1 = 2, h_2 = 0 \in K[\bar{x}]$ s.t. $h_1 \cdot (x+1) + h_2 \cdot (x) = f$. However, if we first choose x as reductor, we get $f \xrightarrow{x} 2$, which cannot be further reduced.*

One solution might be to reduce the generators w.r.t. the other generators. As $x+1 \xrightarrow{x} 1$, $1 \in I$ and we can substitute $x+1$ with 1 in the set of generators. We then could reduce $f \xrightarrow{x} 2 \xrightarrow{1} 0$. An algorithm for reducing generator sets is given in Algorithm 2.

```
1 Input: F = {f_1,...,f_s}
2 Output: G = {g_1,...,g_t} s.t. ⟨f_1,...,f_s⟩ = ⟨g_1,...,g_t⟩
3    and for all i = 1,...,t, g_i cannot be reduced modulo G \ {g_i}.
4 G = F;
5 while (exists g ∈ G with g ≠ ḡ^{G\{g}}) {
6    G = G \ {g};
7    h = ḡ^{G\{g}};
8    if (h ≠ 0) G = G ∪ {h};
9 }
```

Algorithm 2: Ideal reduction.

Simple inter-reducing does not suffice for deciding ideal membership though.

Example 3.2.6. *Let $I = \langle xy+1, y^2-1 \rangle \subset K[x_1, x_2]$. Notice that the generators of I are already reduced. The polynomial $f = xy^2 - x = x(y^2-1)$ is clearly contained in I, but the reduction algorithm may yield the following calculation: $f = y(xy+1) - x - y$ and then the remainder is $-x-y$.*

Here, we could change the order of the polynomials to overcome this. However, we follow another, more general approach, in which we regard the input polynomials as generators for an ideal. If we are able to generate another set of generators for this ideal which have the properties of a unique remainder, we are done.

3.2.3 Foundations of Gröbner bases

We now define Gröbner bases. These bases are sets of generators and allow i.a. easily answering the membership problem. Obviously, in order to have nice properties for the membership-question, the basis has to regard the leading terms of the generators.

Definition 3.2.9. *Let $I \subseteq K[\bar{x}] \setminus \{0\}$. Then $\mathsf{lmc}(I) = \{\mathsf{lmc}(f) : f \in I\}$.*

We can find such generators.

Lemma 3.2.4. *[CLO97, p. 73] Let $I \subseteq K[\bar{x}] \setminus \{0\}$. Then there exist $g_1, \ldots, g_t \in I$ s.t. $\langle \mathsf{lmc}(I) \rangle = \langle g_1, \ldots, g_t \rangle$.*

These generators build a Gröbner basis.

Definition 3.2.10 (Gröbner basis)**.** *For a given monomial order, a finite set $G = \{g_1, \ldots, g_t\} \subset I \subseteq K[\bar{x}]$ is a* Gröbner basis *for I if*

$$\langle \mathsf{lmc}(I) \rangle = \langle g_1, \ldots, g_t \rangle.$$

The existence of a Gröbner basis for an arbitrary ideal can be shown by combining Lemma 3.2.4 and Theorem 3.1.2. The Gröbner basis is also a suitable set of generators for the ideal.

Corollary 3.2.5. *[CLO97, p. 75] If $G \subset I$ is a Gröbner basis, then $\langle G \rangle = I$.*

There are a lot of different characterisations for Gröbner bases. However, the following suffices for our needs. An extended list can be found in [BWK93, p. 206].

Lemma 3.2.6. *[AL94, p. 32] Let I be a non-zero ideal of $K[\bar{x}]$. Let $G \subset I$. Then the following statements are equivalent:*

1. *G is a Gröbner basis.*
2. *$f \xrightarrow{G}{}^{*} 0$ if and only if $f \in I$.*
3. *For all $f \in I$, $f \neq 0$, there exists a $g \in G$ s.t. $\mathsf{lm}(g) \mid \mathsf{lm}(f)$.*

Thus, checking ideal membership is a simple reduction, if we have a Gröbner basis for our ideal.

Definition 3.2.11 (Reduced form)**.** *Let f, f' be polynomials in $K[\bar{x}]$ and I a non-zero ideal of $K[\bar{x}]$. If $G \subset I$ is a Gröbner basis and $f' = \bar{f}^G$, then f' is the* reduced form *of f, written as $\mathsf{red}_G(f)$.*

3.2.4 Constructing a Gröbner basis

The very first algorithm which computed a Gröbner basis was the Buchberger algorithm, proposed by and named after Bruno Buchberger in his PhD. thesis [Buc65]. Most modern algorithms resemble (parts of) this algorithm. In order to understand the correctness of this algorithm, we review the characterisation from Lemma 3.2.6. The important point of a Gröbner basis is that we can guarantee that every leading term of a polynomial in the ideal can be divided by the leading term of a generator. How is that different from arbitrary sets of generators?

Example 3.2.7. *Consider the ideal $I = \langle xy + 1,\ y^2 - 1 \rangle$ (compare Example 3.2.6). Then the polynomial $f = y \cdot (xy+1) - x \cdot (y^2-1) = xy^2 + y - xy^2 + x = x + y$ is in the ideal. However $x + y$ cannot be reduced by the generators of I.*

Here, we see that the leading terms of the two polynomials cancel each other by multiplying each with a suitable monomial. So, if we want to find a Gröbner basis, we have to eliminate the possibility of leading-term cancellation.

We can describe such cancellations with *S-polynomials*, which we define next.

Definition 3.2.12 (Least common multiple)**.** *For two monomials s,$t \in M_{\bar{x}}$, and their exponent vectors $\alpha = (\alpha_1 \ldots,\alpha_n)$ and $\beta = (\beta_1,\ldots,\beta_n)$, let the exponent vector $\gamma = (\gamma_1,\ldots,\gamma_n)$ be defined by $\gamma_i = \max(\alpha_i,\beta_i)$ for all $1 \leq i \leq n$. The* least common multiple *of s and t is then defined as $\mathsf{lcm}(s,t) = \bar{x}^{\gamma}$.*

Definition 3.2.13 (S-polynomial)**.** *Assume f,$g \in K[\bar{x}]\backslash\{0\}$. The* S-polynomial *of f and g is defined as*

$$S(f,g) = \frac{\mathsf{lcm}(\mathsf{lm}(f),\mathsf{lm}(g))}{\mathsf{lmc}(f)} \cdot f - \frac{\mathsf{lcm}(\mathsf{lm}(f),\mathsf{lm}(g))}{\mathsf{lmc}(g)} \cdot g.$$

The S-polynomial is not only the easiest way to produce cancellation, it is even sufficient to provide all cancellations [CLO97, p.81]. Therefore, our main ingredient for constructing a Gröbner basis is finding cancellation by S-polynomials, and if all S-polynomials are already in the ideal, then we have found a basis.

Theorem 3.2.7 (Buchberger criterion)**.** *[CLO97, p. 83] Let $G = \{g_1,\ldots,g_t\} \subset K[\bar{x}] \setminus \{0\}$ be a set of generators and $I = \langle G\rangle$ its ideal. Then G is a Gröbner basis if and only if $S(g_i,g_j) \xrightarrow{G}{}^{*} 0$ for all $1 \leq i < j \leq t$.*

This immediately gives rise to the Buchberger algorithm (Algorithm 3). The

```
1  Input: F = {f_1,...,f_r}, ≥;
2  Output: G Gröbnerbasis w.r.t. ≥ s.t. ⟨F⟩ = ⟨G⟩;
3  G = F;
4  S = {(f_i,f_j) : 1 ≤ i < j ≤ r};
5  while(S ≠ ∅) {
6     select (f_1,f_2) ∈ S;
7     S = S \ {(f_1,f_2)};
8     q = S(f_1,f_2);
9     q = Reduce(q, G, ≥);
10    if(q ≠ 0) {
11       S = S ∪ {(f_i,s) : f_i ∈ G};
12       G = G ∪ {q};
13    }
14 }
```

Algorithm 3: Buchberger algorithm.

reduce-procedure does a (full) reduction (Algorithm 1). The correctness proof is based on Theorem 3.2.7 and the fact that the algorithm checks for cancellations between all elements in the basis. Whenever a new element is added, all new pairs are also scheduled. A termination proof would show that if we would keep on adding elements, we eventually have to add the 1, and afterwards, no new elements could be added since the reduction would always reduce to zero [BWK93, p. 213].

Example 3.2.8. *We give an example calculation. We calculate the Gröbner basis w.r.t. the graded-lexicographic order for the ideal $I = \langle f_1,f_2\rangle$ with $f_1 = xy + 1$ and $f_2 = y^2 - 1$ from Example 3.2.6. We initialize and set $G = \{f_1, f_2\}$ and $S = \{(f_1,f_2)\}$.*

1. *We select* $(f_1,f_2) \in S$. $q = S(f_1, f_2) = \frac{xy^2}{xy} \cdot (xy+1) - \frac{xy^2}{y^2}(y^2-1) = x+y$. $x+y$ *cannot be reduced by the elements in* G. *Thus,* $G = \{f_1, f_2, f_3\}$ *and* $S = \{(f_1,f_3), (f_2,f_3)\}$ *with* $f_3 = x+y$.
2. *We select* $(f_1,f_3) \in S$. $q = S(f_1, f_3) = \frac{xy}{xy} \cdot (xy+1) - \frac{xy}{x} \cdot (x+y) = -y^2+1$. $q \xrightarrow{f_2} 0$.
3. *We select* $(f_2,f_3) \in S$. $q = S(f_2, f_3) = \frac{xy^2}{y^2} \cdot (y^2-1) - \frac{xy^2}{x} \cdot (x+y) = -y^3-x$. $q \xrightarrow{f_2} -x-y \xrightarrow{f_3} 0$.

We are done. The Gröbner basis is $\{xy+1,\ -y^2+1,\ x+y\}$.

There are several improvements to this Buchberger algorithm. We discuss them in Chapter 4. We now return to the remaining problem of finding a small generating set.

3.2.5 Reduced Gröbner basis calculation

Gröbner bases give us a nice set of generators, but they're neither minimal nor unique. Minimality is obviously important to us, as we might want to pass the Gröbner basis during our SMT procedure. Uniqueness is useful for comparison. Luckily, we get uniqueness for free if we minimize and reduce our Gröbner basis.

Definition 3.2.14 (Reduced Gröbner basis)**.** *Let* G *be a Gröbner basis for an ideal* I *such that for all polynomials* $f \in G$*:*

1. $\mathsf{lc}(f) = 1$.
2. *For all monomials* m *which appear in* f *with a non-zero coefficient, the monomial* m *does not lie in* $\langle \mathsf{lm}(G \setminus \{f\}) \rangle$.

Then G *is a* reduced Gröbner basis.

The straightforward algorithm for this is given in Algorithm 4 and taken from [BWK93, p 216]. If we sort F, then we only have to check against elements of H.

Example 3.2.9. *Consider the Gröbner basis* $G = \{xy+1,\ y^2-1,\ x+y\}$ *from Example 3.2.8. Then the reduced Gröbner basis is obtained by removing* $xy+1$, *because* $\mathsf{lm}(x+y) \mid \mathsf{lm}(xy+1)$. *No further steps have to be taken, because neither of the monomials in the remaining polynomials can be reduced by a leading monomial of the other polynomials.*

Lemma 3.2.8. *[CLO97, p. 90] For any ideal* $I \neq \{0\}$*, and some given variable orders and monomial orders,* I *has a unique Gröbner basis.*

We will use reduced Gröbner bases as generators for ideals.

3.3 The Nullstellensatz

We are now able to calculate a 'nice' set of generators, and can easily decide ideal membership. But we are more interested in the corresponding varieties. In Section 3.1.3 we showed the strong correspondence. In this section, we consider three further questions.

```
1  Input : G Gröbner basis;
2  Output : H reduced Gröbner basis s.t. ⟨G⟩ = ⟨H⟩;
3  H = ∅;
4  F = sort(G); // ascending, by G's monomial order
5  i = 1;
6  while ( i <= size(F) ) {
7    j = 1; div = false;
8    while ( j <= size(H) ) {
9      if(lm(h_j) | lm(f_i)) div = true;
10   }
11   if(!div) H = H ∪ {f_i/lc(f_i)};
12 }
13 H = Reduction(H);
```

Algorithm 4: Reduced Gröbner basis.

Question 3. *How can we decide if for a given ideal I it holds that $V_{\mathbb{C}}(I) = \emptyset$?*

This question is answered by the Weak Nullstellensatz, which is discussed in Section 3.3.1.

Question 4. *How can we decide if for a given ideal I with $V_{\mathbb{C}}(I) \neq \emptyset$ it holds that $V_{\mathbb{R}}(I) = \emptyset$?*

Notice that $V_{\mathbb{C}}(I) = \emptyset$ implies $V_{\mathbb{R}}(I) = \emptyset$. The question is answered by the Real Nullstellensatz, which is discussed in Section 3.3.2.

Since we are only interested in applying the theory to real algebra, we consider the field $\mathbb{R}$, the algebraically closed extension $\mathbb{C}$ and the countable subset $\mathbb{Q}$. First we formalise some of their properties.

Proposition 3.3.1. *[BWK93, p. 306] The complex numbers are algebraically closed, i.e., each non-constant polynomial in $\mathbb{C}[\bar{x}]$ has a zero.*

The reals are not algebraically closed, for instance $x^2 + 1$ has no real zero.

Definition 3.3.1 (Sum of squares)**.** *Assume polynomials $p_1, \ldots, p_n \in K[\bar{x}]$. Then $\sum_{i=1}^{n} p_i^2$ is a* trivial sum of squares *(TSQ). If a polynomial p can be rewritten as a trivial sum of squares q, $p = q$, then p is a* sum of squares*.*

Definition 3.3.2 (Real field)**.** *A field K is* real *if and only if $-1 \in K$ is not a sum of squares.*

Proposition 3.3.2. *The rational numbers and the real numbers are both real fields. The complex numbers are not a real field.*

3.3.1 Weak Nullstellensatz

From Corollary 3.1.4 we know that $1 \in I \implies V(I) = \emptyset$. However, the other direction also holds.

Theorem 3.3.3 (Weak Nullstellensatz)**.** *[CLO97, p. 168] Let $I \subset \mathbb{C}[\bar{x}]$ be an ideal. Then $V_{\mathbb{C}}(I) = \emptyset$ implies $1 \in I$.*

This is a direct consequence from a famous result by Hilbert -the Hilbert Nullstellensatz.

From this theorem and Corollary 3.1.4 we get the following result:

Corollary 3.3.4. *Let $I \subset \mathbb{C}[\bar{x}]$ be an ideal and G the reduced Gröbner basis of I. Then $V_{\mathbb{C}}(I) = \emptyset$ if and only if $G = \{1\}$.*

Thus by calculating the Gröbner basis for a given set of polynomials, we can easily decide if these polynomials have common zeroes.

3.3.2 Real Nullstellensatz

In this part, we consider input sets which are consistent over the complex numbers. We first characterise real fields and use this observation to state a very intuitive but strong theorem.

Definition 3.3.3. *Given a polynomial p over a real field R, then p is:*

- positive definit *if $p(\bar{a}) > 0$ for all $\bar{a} \in R^n$.*
- positive semidefinite *if $p(\bar{a}) \geq 0$ for all $\bar{a} \in R^n$.*

When considering the reals, we know that any square is positive. The following are some results from a formal treatment in [BPR06, Chapter 2].

Proposition 3.3.5. *Given a real field R, the following statements hold:*

1. *$\{a \cdot a : a \in R\} \subseteq \{a : a \geq 0\}$ and $\{a^2 : a \in \mathbb{R}\} = \{a : a \geq 0\}$.*
2. *Given a polynomial $p \in R[\bar{x}]$, p^2 is positive semidefinite.*
3. *Sums of squares are positive semidefinite.*

Note that not all positive semidefinite polynomials are sums of squares, famous examples can be found in [Mot67].

Next we consider the *Real Nullstellensatz*, which is due to Stengle [Ste74]. A more general form including also inequalities is called the *Positivstellensatz*. Here we give the version of the Real Nullstellensatz from [Har07].

Theorem 3.3.6 (Real Nullstellensatz)**.** *Let R be a real field. Given polynomials $p_1, \ldots, p_n \in R[\bar{x}]$ with $I = \langle p_1, \ldots, p_n \rangle$ and $V_{\mathbb{C}}(I) \neq \emptyset$, then $V_{\mathbb{R}}(I) = \emptyset$ if and only if there exists a polynomial q, which is a TSQ such that $1 + q \in I$.*

3.4 Handling inequalities

In Chapter 2 we considered systems of equalities and inequalities. We now show two ways to handle them. For a complete decision procedure, we will transform the inequalities into equivalent equalities. For simplification, we show that we can indeed reduce inequalities by a Gröbner basis.

Proposition 3.4.1. *[PQR09, Proposition 1] The following equivalences hold:*

1. *$p \geq 0 \iff \exists y. p - y^2 = 0$.*
2. *$p \neq 0 \iff \exists y. py - 1 = 0$.*

3. $p > 0 \iff \exists y.py^2 - 1 = 0.$

The proof uses two properties, namely squares are exactly the positive numbers in $\mathbb{R}$ and only the zero has no multiplicative inverse element.

This transformation comes at the cost of introducing several new variables. Therefore, we are also interested in just reducing inequalities.

Proposition 3.4.2. *[DS97b, Proposition 4.1] Given a formula φ with relations $\sim \in \{>, \geq, <, \leq, \neq\}$ of the following form:*

$$\varphi = \bigwedge_{i=1}^{s} p_i = 0 \wedge \bigwedge_{i=1}^{t} q_i \sim 0.$$

Then for the Gröbner basis $G = \{g_1, \ldots, g_m\}$ of $I = \langle p_1, \ldots, p_s \rangle$ the following holds:

$$\varphi \implies \bigwedge_{i=1}^{m} g_i = 0 \wedge \bigwedge_{i=1}^{t} \mathsf{red}_G(q_i) \sim 0.$$

Proof. That the substitution of the equalities by a Gröbner basis is equisatisfiable follows from, e.g., Corollary 3.2.5. We only need to show that $g_i = 0$ for all i implies $q_i \sim 0$ if and only if $\mathsf{red}_G(q_i) \sim 0$. For each q_i there exist suitable h_j with $1 \leq j \leq m$ such that $q_i = h_1 g_1 + \ldots + h_m g_m + \mathsf{red}_G(q_i)$ by definition. Thus, $q_i(\bar{a}) = h_1 g_1(\bar{a}) + \ldots + h_m g_m(\bar{a}) + \mathsf{red}_G(q_i)(\bar{a})$. By our precondition -all g_i evaluate to 0- we only need to consider the set $S = \{\bar{a} \in \mathbb{R}^n : g_i(\bar{a}) = 0, 1 \leq i \leq m\}$. On S, all terms $h_i g_i$ thus evaluate to zero. Hence $q_i(\bar{a}) = \mathsf{red}_G(q_i)(\bar{a})$ for all $\bar{a} \in S$. □

Chapter 4

The SMT-RAT Gröbner basis module

In this chapter, we first show how we wrap a Gröbner basis calculation into a module in the SMT-RAT framework and how we make this module SMT-compliant. Afterwards we optimize the standard Buchberger algorithm by introducing improved algorithms and data structures for calculating Gröbner bases.

4.1 An SMT module based on Gröbner basis calculation

We start the description with a general overview of the behaviour of a minimal implementation of a module, which uses Gröbner basis computations for *simplification*.

Whenever a constraint is added, the module checks whether the received constraint is an equality. If it is an inequality, the module adds the inequality to the passed formula, such that backend modules handle it. If it is an equality, it is added to a set of generators for an ideal I.

As soon as the module's consistency check is called, the Gröbner basis of I is calculated. If the Gröbner basis equals $\{1\}$, then `unsat` is returned. Otherwise the Gröbner basis is passed to the backend modules, and then the backend modules are called. Notice that if a new Gröbner basis is passed, the module removes the old Gröbner basis from the passed formula. As reason set for the passed polynomials from the Gröbner basis, the set of generators is used.

The remainder of this section gives more details on the behaviour and some of the improvements made to the module.

4.1.1 Incrementality and backtracking

Changing the Buchberger algorithm to support incremental calls is rather natural. First, we notice that for a given set $I = \{p_1, \ldots, p_n\}$, its Gröbner basis $G = \{g_1, \ldots, g_m\}$ and a polynomial q, we have equality of ideals $\langle I \cup \{q\}\rangle = \langle G \cup \{q\}\rangle$. Second, if we want to calculate the Gröbner basis of $\langle G \cup \{q\}\rangle$, we only have to calculate the S-polynomials for the pairs (g_i, q). All other pairs reduce to

zero because G is a Gröbner basis. We formalise the incremental Buchberger algorithm in the next section (Algorithm 5).

Gröbner bases in the module are represented by objects, which save the already calculated Gröbner basis as well as a list of additional polynomials which have been added to the object since the last consistency check, but are not yet part of the Gröbner basis, as the consistency check has not been called after adding these polynomials (see Figure 4.1).

new:	GB:
{ }	$\{x,\ y+z\}$

(a) Before: the GB is $\{x,\ y+z\}$.

new:	GB:
$\{y\}$	$\{x,\ y+z\}$

(b) The constraint $y = 0$ is asserted.

new:	GB:
{ }	$\{x,\ y,\ z\}$

(c) After the consistency check, we have a new GB.

Figure 4.1: The Gröbner basis object while adding the equality $y = 0$.

Removing generators from the ideal is more involved, because calculating this directly entails the computation of an intersection, resulting in a Gröbner basis calculation with additional variables [BWK93, Section 6.2]. Instead of this, we propose an approach inspired by the chronological backtracking in DPLL (the latter is explained in e.g. [KS08]).

We implement the chronological backtracking by introducing a stack with Gröbner basis calculation objects. After a Gröbner basis calculation, the topmost element of the stack is updated, such that the additional polynomials are considered in the newly calculated Gröbner basis. After removing an equality, the elements are popped from the stack until we are before the point where the now removed equality was added. In the next step, all equalities which were added afterwards are added again (see Figure 4.2).

This approach might lead to an almost new Gröbner basis calculation, but as the currently used SAT solver mostly uses chronological backtracking, the chance of frequently removing equalities which were added early during the incremental theory calls seems small.

{ }	$\{x,y,z\}$
{ }	$\{x,y\}$
{ }	$\{x\}$

(a) Before: the equalities $x = 0$, $y = 0$, $z = 0$ were added.

{ }	$\{x\}$

(b) Everything calculated after $y = 0$ was added is removed.

$\{z\}$	$\{x\}$
{ }	$\{x\}$

(c) The equality $z = 0$ is scheduled as new input again.

Figure 4.2: The Gröbner basis object stack while removing the equality $y = 0$.

4.1.2 Generating smaller conflict subsets

If the Gröbner basis equals $\{1\}$, the set of all equalities is inconsistent. However, in most of these infeasible cases a subset of the equalities would already yield $\{1\}$ as its Gröbner basis.

To determine such a subset, de Moura and Passmore [dMP09] introduced certificates for inconsistency. It was also shown that minimality of these certificates is a problem which is as hard as calculating the Gröbner basis.

These certificates are basically tuples of polynomials $(h_1, \ldots, h_n)$ such that for an ideal $I = \langle f_1, \ldots, f_n \rangle$ and a polynomial $p \in I$ we have $\sum_{i=1}^{n} h_i f_i = p$ for suitable $h_i \in K[\bar{x}]$. In the case of inconsistency, we have $p = 1$.

Calculating these certificates requires the reductions within the Gröbner basis calculation to be extended to ordinary divisions, which is certainly less efficient. In SMT-RAT, we pass the Gröbner basis to other modules, and we have to provide reason sets. Smaller reason sets yield smaller conflict sets if a conflict is found in a successor module. The calculation of small reason sets requires the computation of certificates for each element in the Gröbner basis.

We try to provide small conflict and reason sets in a more naive way. During the Gröbner basis calculation, we attach a bitvector to the polynomials. For an input ideal $I = \langle f_1, \ldots, f_n \rangle$ and an arbitrary polynomial p which appears during the calculation, the bitvector $b(p) = (b_1, \ldots, b_n)$ has the semantics that $b_i = 1$ if and only if p_i was used somewhere to construct p. This way, after the computation of the Gröbner basis for I, we can extract an over-approximation of the reason set for each element in the Gröbner basis, and thus, if $1 \in I$, a smaller conflict set.

Example 4.1.1. *We calculate the reduced Gröbner basis for the ideal* $I = \langle f_1,\ f_2,\ f_3 \rangle$, *with* $f_1 = x$, $f_2 = y^2 + z$ *and* $f_3 = y$. *The S-polynomial from* f_2, f_3 *yields after a reduction with* f_3 *the polynomial* $f_4 = z$. *The reduced Gröbner basis is then* $\{f_1, f_3, f_4\}$. *The bitvectors for* f_1,f_2,f_3 *are clear:* $b(f_1) = 100$, $b(f_2) = 010$, $b(f_3) = 001$. *The bitvector of* f_4 *is the bitwise-or of* $b(f_2)$ *and* $b(f_3)$, *which yields* $b(f_4) = 011$. *This means that* $z = 0$ *follows from* $y^2 + z = 0$ *and* $y = 0$, *which is true.*

We formalise this approach by the next theorem.

Definition 4.1.1 (Reason vector)**.** *For a given ideal* $I = \langle f_1, \ldots, f_n \rangle$ *the* reason vector $b(p) = (b_1, \ldots, b_n)$ *of* $p \in I$ *is recursively defined as follows:*

1. *If* $p = f_i$ *for some* $1 \leq i \leq n$, *then* $b_i(p) = 1$ *and* $b_j(p) = 0$ *for all* $j \neq i$.

2. *If* $p = S(q_1, q_2)$ *then* $b_i(p) = 1 \iff b_i(q_1) = 1 \vee b_i(q_2) = 1$ *for* $q_1, q_2 \in I$.

3. *If* $p' \xrightarrow{q} p$ *then* $b_i(p) = 1 \iff b_i(q) = 1 \vee b_i(p') = 1$ *for* $p', q \in I$.

Theorem 4.1.1. *If* b *is a reason vector for* p *during the computation of the Gröbner basis for* $I = \langle f_1, \ldots, f_n \rangle$ *then* b *is well-defined and* p *can be written as* $\sum_{i=1}^{n} b_i h_i f_i$ *with suitable* $h_i \in K[\bar{x}]$.

Proof. The vector b is well-defined, as the Buchberger algorithm only consists of a chain of the operations mentioned above. We prove the correctness by induction, and distinct the following three cases:

1. $p = f_i$ for some $1 \leq i \leq n$ then $p = 1 \cdot 1 \cdot f_i$.

2. $p = S(q_1, q_2)$ for some $q_1, q_2 \in I$ then

$$\begin{aligned} p &= m_1 q_1 + m_2 q_2 \\ &= (m_1 \sum_{i=1}^{n} b_i(q_1) \cdot h_{1i} \cdot f_i) + (m_2 \sum_{i=1}^{n} b_i(q_2) \cdot h_{2i} \cdot f_i) \\ &= \sum_{i=1}^{n} b_i(p) \cdot (b_i(q_1) \cdot m_1 h_{1i} + b_i(q_2) \cdot m_2 h_{2i}) \cdot f_i \end{aligned}$$

   Notice that the multiplication with $b_i(p)$ does not cancel non-zero terms, because if a term is non-zero, then either $b_i(q_1) = 1$ or $b_i(q_2) = 1$, and then $b_i(p) = 1$.

3. $p' \xrightarrow{q} p$ for some p', q, then

$$\begin{aligned} p &= p' + mq \\ &= (\sum_{i=1}^{n} b_i(p') \cdot h_{1i} \cdot f_i) + (m \sum_{i=1}^{n} b_i(q) \cdot h_{2i} \cdot f_i) \\ &= \sum_{i=1}^{n} b_i(p) \cdot (b_i(p') \cdot h_{1i} + b_i(q) \cdot m h_{2i}) \cdot f_i \end{aligned}$$

   with the same argument as applied in case 2.

□

Instead of using all equalities as reason/conflict set, the module extracts the set from the Gröbner basis elements. We extended the calculations of the S-polynomial and the calculation according to the Definition 4.1.1.

4.1.3 Handling inequalities

To handle inequalities, the module offers the possibilities described in Section 3.4. The transformation approach is a direct implementation of the equivalences from Proposition 3.4.1. For each inequality in the set of received constraints, we generate a fresh variable. Then, we handle the transformed constraint as we handled equalities before.

Instead of transforming the inequalities into equalities, in the simplification approach we just reduce the inequalities with respect to our Gröbner basis, as in Proposition 3.4.2. To make this SMT-compliant, we introduce an *inequalities table*. In this table, each received inequality corresponds to a row. The row has the memory location of the original (received) inequality as index. Each cell has a number (the *backtrack number*) which identifies with which Gröbner basis the equality was reduced, and a polynomial which represents the reduced polynomial.

Newly added inequalities are only reduced w.r.t. the latest Gröbner basis. If an inequality c is removed, then the corresponding row is erased. If an equality is removed, all cells with a backtrack number greater then b are removed, where b is the number of equalities which were added before c.

Example 4.1.2. *For the following constraints with their memory addresses* $0x1 : x = 0$, $0x2 : x + y \geq 0$, $0x3 : y + z \geq 0$, $0x4 : y = 0$, $0x5 : z = 0$ *and* $0x6 : x + z$ *which were successively added to our module, we get the following stack with Gröbner bases:*

$$(0 : \{\};\ 1 : \{x\};\ 2 : \{x,y\};\ 3 : \{x, y, z\})$$

The inequalities table then looks like this:

0x2	*(0)* $x + y$	*(1)* y	*(2)* 0
0x3	*(0)* $y + z$	*(2)* z	*(3)* 0
0x6	*(0)* $x + z$	*(3)* 0	

We regard the first row, as the argumentation for the other rows is analogous. The memory address gives a reference to the constraint. The first cell is the original polynomial, which is always stored. During the theory call, the inequality is reduced to y*. The number of equalities which were added to the Gröbner basis is* 1*, so the backtrack number is* 1*. After another equality has been added, with the next theory call, we can reduce the polynomial further. This yields the polynomial* 0*, and* 2 *as backtrack number. Notice that there is no cell with backtrack number* 3*, as the polynomial* 0 *cannot be reduced further w.r.t. the third Gröbner basis.*

With the reduction of inequalities, we may reduce inequalities to zero or to constants, which can be replaced by one of the Boolean values. If an inequality $p \sim 0$ reduces to $c \sim 0$, we have to distinct some cases, displayed in Figure 4.3.

	$\sim = \leq$	$\sim = \geq$	$\sim = <$	$\sim = >$	$\sim = \neq$
$c = 0$	`true`	`true`	`false`	`false`	`false`
$c > 0$	`false`	`true`	`false`	`true`	`true`
$c < 0$	`true`	`false`	`true`	`false`	`true`

Figure 4.3: Different cases after reducing a polynomial to a constant.

Reduction to `false` yields a conflict, and the module either returns directly from its consistency check, or continues and looks for more conflicts with inequalities, such that it might return a bunch of conflict sets at once.

Reduction to `true` makes the constraint superfluous, which follows directly from the fact that our set of constraints is a conjunction, so there is no reason to pass it to the backend modules. Moreover, this information can be shared with the predecessors of the current module by a theory deduction.

Generation of conflict and reason sets is done similarly as before. The reduced polynomial's reason vector is extracted, but this time, the module adds the original inequality to the reason set afterwards, as it is not taken into account in the reason vector during the reduction.

Notice that the module is thus only capable of finding conflicts which are caused by a set of equalities, together with at most one inequality.

4.2 Improving the Buchberger algorithm

We revisit the Buchberger algorithm (see Algorithm 3). From experimental results (e.g. in [RS] and [Fau02]) we know that the number of superfluous S-

polynomials is usually very high. We discuss two criteria which indicate that a S-polynomial is superfluous, given by Buchberger (e.g. [Buc79]).

We then briefly discuss some strategies on how to choose the S-polynomial to process next, as this order has a strong influence on the computation time. Afterwards we give the implementation which is based on an idea from Gebauer and Möller [GM88]. As the reduction is the main procedure during the algorithm, we conclude with a short discussion on some strategies to find suitable reductors to reduce the S-polynomial fast.

To streamline our argumentation, we introduce *S-pairs*.

Definition 4.2.1 (S-pair)**.** *A pair of two polynomials f,g for which the S-polynomial has to be regarded during the calculation of a Gröbner basis is called an* S-pair*. We call an S-pair* superfluous *if the S-polynomial of f,g reduces to zero. Otherwise we call an S-pair* critical*.*

4.2.1 Criteria for superfluous S-pairs

We start with an example of an S-polynomial which reduces to zero.

Example 4.2.1. *Let $I = \langle f, g\rangle$ with $f = x^2 + 2y + 2$ and $g = y^2 + y$. Notice that $\mathsf{lcm}(f,g) = fg$. Then*

$$\begin{aligned} S(f,g) &= y^2 \cdot f + -x^2 \cdot g \\ &= y^2 \cdot (2y+2) + -x^2 \cdot (y) \\ &\xrightarrow{f} y^2 \cdot (2y+2) + (2y+2) \cdot (y) \\ &= g \cdot (2y+2) \\ &\xrightarrow{g}{}^{*} 0 \end{aligned}$$

In the example above, notice that the leading monomials are disjoint. The resulting S-polynomial can be reduced to zero by the original polynomials. Disjoint leading monomials are a sufficient criterion for an S-pair to be superfluous, which is formalised by the next lemma.

Lemma 4.2.1 (Buchberger's first criterion)**.** *[BWK93, Lemma 5.66] Let $f,g \in K[\bar{x}]$. If $\mathsf{lcm}(\mathsf{lm}(f),\mathsf{lm}(g)) = \mathsf{lm}(f)\mathsf{lm}(g)$, then $S(f,g) \xrightarrow{f,g}{}^{*} 0$.*

The proof is a simple generalisation of the example above. Checking this criterion is rather easy, as it involves only the two polynomials in the S-pair.

The description of the second criterion is more complex, as it is a global criterion stating that although an S-pair may be critical at the current state of the calculation, it will be superfluous once we have reduced other S-polynomials. The description follows [BWK93], in which a modified argument is given, such that we do not have to introduce several algebraic notions. From the original argumentation, e.g. in [CLO97], one might extract the following intuition:

The function of S-polynomials is to provide all possible cancellations between elements of the Gröbner basis. Let us assume that the cancellations itself are an ideal, and regard the S-polynomials as generators for this 'cancellation ideal'. Certain generators may be a product of other generators, so they can be eliminated from the set of generators.

Definition 4.2.2 (T-representation)**.** *Let $F \subset K[\bar{x}]$ be a set of polynomials. Let $0 \neq f = \sum_{i=1}^{l} m_i f_i$ with $m_i \in M_{\bar{x}}$ and $f_i \in F$ for all $1 \leq i \leq l$. If $\max\{\mathsf{lm}(m_i f_i)\} \leq t$ for some $t \in M_{\bar{x}}$ then f is in t-*representation *with respect to F.*

Theorem 4.2.2. *[BWK93, Theorem 5.64] Let G be a finite subset of $K[\bar{x}] \setminus \{0\}$. If for all $g_1, g_2 \in G$ with $s = S(g_1, g_2)$ it holds that either $s = 0$ or s has a t-representation with $t < \mathsf{lcm}(\mathsf{lm}(g_1), \mathsf{lm}(g_2))$, then G is a Gröbner basis.*

Intuitively, if an S-polynomial can be represented by (other) elements in G in a way which only causes 'smaller cancellations' than the biggest cancellation produced by the S-polynomial, then this S-polynomial is superfluous. In terms of the 'cancellation ideal', the cancellation by the S-polynomial can be generated by other cancellations. A description for this case is refined by the second criterion.

Lemma 4.2.3 (Buchberger's second criterion)**.** *[BWK93, Proposition 5.70] Let G be a finite subset of $K[\bar{x}] \setminus \{0\}$ and f, g_1, g_2 such that:*

1. $\mathsf{lm}(f) \mid \mathsf{lcm}(\mathsf{lm}(g_1),\mathsf{lm}(g_2))$
2. *$S(g_i,p)$ has a t_i-representation w.r.t. G with $t_i < \mathsf{lcm}(\mathsf{lm}(g_i), \mathsf{lm}(p))$ and $i \in \{1,2\}$:*

Then $S(g_1, g_2)$ has a t-representation w.r.t. G for a $t < \mathsf{lcm}(\mathsf{lm}(g_1), \mathsf{lm}(g_2))$.

Since the correctness of our algorithm does not depend on the order in which the S-pairs are treated, we could postpone the reduction of the polynomial $S(g_1, g_2)$ to the end. Then, at the point where the S-pair (g_1, g_2) is the only remaining pair, we already have a Gröbner basis by Theorem 4.2.2. It follows directly that $S(g_1, g_2)$ is superfluous.

Before we refine how to apply the criteria, we discuss strategies for selecting the S-polynomial.

4.2.2 Strategies for choosing the next S-pair

In the discussion of the Buchberger algorithm, we said that the an arbitrary S-pair can be selected for treatment, but the efficiency of the algorithm is gravely influenced by the selection strategy. From Buchberger originates the idea to take the S-pair with the smallest least common multiple [Buc85]. The corresponding S-polynomial tends to yield non-zero remainder after reduction, and thus we reach the final Gröbner basis elements faster. Thereby the number of additional S-pairs that have to be checked decreases. This strategy is often called the *normal strategy.*

Definition 4.2.3 (Homogeneous polynomial)**.** *A polynomial $p = \sum_{i=1}^{l} a_i m_i$ with $a_i \in K \setminus \{0\}$ and $m_i \in M_{\bar{x}}$ for all $1 \leq i \leq l$ is called* homogeneous *if there exists a $c \in K$ such that $\mathsf{tdeg}(m_i) = c$ for all $1 \leq i \leq l$.*

The normal strategy performs quite well, provided the ordering is a degree-ordering. Moreover, the performance for homogeneous input polynomials is far better with this strategy and a slightly modified Buchberger algorithm, as described in [BWK93, Chaper 10]. Making the input homogeneous before the

calculation and dehomogenising it afterwards tends to construct unnecessarily big, intermediate Gröbner bases. This is improved by a further refinement, called *sugar strategy*, which was proposed by Giovini et al. in [GMN+91]. The sugar strategy adds 'sugar' to the polynomials in order to emulate homogeneity, and emulates the selection-order of S-pairs which would be taken if the input was indeed homogeneous. This approach is further improved by Bigatti et al. in [BCR11].

4.2.3 The Gebauer-Möller variant

We now give the algorithm as we implemented in GiNaCRA. Besides the criteria for superfluous S-pairs, we also made a couple of other changes to the basic algorithm. The basis for this algorithm was taken from [BWK93, Algorithm GRÖBNERNEW2], although the algorithm is originally presented in [GM88]. We modified it such that it:

1. works incrementally as necessary for SMT-compliance,
2. reduces the input polynomials,
3. returns immediately if an S-polynomial is reduced to 1.

On a technical level, we changed the S-pairs, such that they now save indices instead of polynomials or references to polynomials. The proof for correctness and termination in its basic form can be found in [BWK93, Theorem 5.73]. Here, we give some intuition on what happens. Currently, the implementation uses the normal strategy, but the algorithm is also correct with another order to pick the next S-pair.

Let us first assume that `updatePairs(G,S,p)` adds pairs (p,g) with $g \in G$ to the set of critical pairs S.

In the first part of the algorithm, we simplify the polynomials with respect to the other generators, as in Algorithm 2 and then, in the second part, we add the polynomials one by one to our Gröbner basis. By a call to `updatePairs` we extend the set of critical pairs accordingly. This makes the algorithm incremental, as we never add a pair which has already been considered by the algorithm.

Returning $\{1\}$ if we find that 1 is in the ideal lets the Buchberger algorithm skip reducing or eliminating S-pairs.

Now we take a closer look at `updatePairs` (Algorithm 6). The description follows [BWK93]. We start with the notion of Buchberger triples.

Proposition 4.2.4. *Given h, g_1, $g_2 \in K[\bar{x}] \setminus \{0\}$. The following statements are equivalent:*

- $\mathsf{lm}(h) \mid \mathsf{lcm}(\mathsf{lm}(g_1), \mathsf{lm}(g_2))$
- $\mathsf{lcm}(\mathsf{lm}(h), \mathsf{lm}(g_1)) \mid \mathsf{lcm}(\mathsf{lm}(g_1), \mathsf{lm}(g_2))$
- $\mathsf{lcm}(\mathsf{lm}(h), \mathsf{lm}(g_2)) \mid \mathsf{lcm}(\mathsf{lm}(g_1), \mathsf{lm}(g_2))$

Definition 4.2.4 (Buchberger triple)**.** *Given h, g_1, $g_2 \in K[\bar{x}] \setminus \{0\}$. If the statements from Proposition 4.2.4 hold, then $(h; g_1, g_2)$ is called a* Buchberger triple*. A Buchberger triple is said to be* strict*, if we do not have equality in the statements from 4.2.4.*

```
1  Input: Set of polynomials P, and Gröbner basis F;
2  Output: Gröbner basis G s.t. ⟨G⟩ = ⟨P ∪ F⟩;
3  I = F;
4  J = ∅;
5  sort(I); //w.r.t. monomial order on the leading monomials.
6  foreach (p ∈ P) {
7     q = Reduce(p,I);
8     if( q ≠ 0 ) { I.add(q); J.add(q); )
9  }
10 G = F;
11 S = ∅; // set of S-pairs
12 foreach (p ∈ J) {
13    J = J \ {p};
14    (G,S) = updatePairs(G,S,p);
15 }
16 while( S ≠ ∅ ) {
17    select s ∈ S;
18    S = S \ {s};
19    q = reduce(S-pol(s),G);
20    q = q/lc(q);
21    if(q == 1) return {1};
22    if(q ≠ 0) (G,S) = updatePairs(G,S,q);
23 }
24 return reduce(G);
```

Algorithm 5: `Buchberger::calculate`: Implementation of Buchberger algorithm in GiNaCRA.

Notice that in a Buchberger triple, the second and third polynomials can be switched, but the first position has a distinct meaning.

By the second Buchberger criterion (Proposition 4.2.3) we know that if $(h; g_1, g_2)$ is a Buchberger triple and the pairs (h, g_1) and (h, g_2) have been considered by the Buchberger algorithm, then the pair (g_1, g_2) is superfluous. We can even eliminate the pair before (h, g_1) and (h, g_2) have been considered, but then, we have to make sure we do not eliminate two pairs from the same triple.

Example 4.2.2. *Let* $h, g_1, g_2 \in K[\bar{x}] \setminus \{0\}$. *Assume* $\mathsf{lcm}(\mathsf{lm}(g_1), \mathsf{lm}(h)) = \mathsf{lcm}(\mathsf{lm}(g_1), \mathsf{lm}(g_2))$. *Then* $(h; g_1, g_2)$ *as well as* $(g_2; h, g_1)$ *are Buchberger triples. Now we could eliminate two pairs:* (g_1, g_2) *on account of the first triple, and* (h, g_1) *on account of the second triple.*

Hence, to prevent the algorithm from eliminating two pairs based on the same set of three polynomials, it should only eliminate pairs before treatment of the other pairs if we have a strict Buchberger triple.

The algorithm starts generating all S-pairs (g, q) with $g \in G$. From these, we first eliminate those pairs which are within a Buchberger triple. Since we just added these pairs, there is no possibility that we used them in eliminating other pairs. Afterwards we remove pairs with disjoint leading monomials. In

```
1  Input: Set of polynomials G, set of critical pairs S, polynomial q;
2  Output: Set of polynomials G, set of critical pairs S;
3  B = ∅;
4  foreach (p ∈ G) {
5    B = B ∪ (p, q);
6  }
7  foreach (s ∈ B) {
8    if (s part of a Buchberger triple in B and s not prime )
9      B = B \ {s};
10 }
11 foreach ((p,p′) ∈ B) {
12   if (pp′ = lcm(p,p′))
13     B = B \ {p,q};
14 }
15 foreach ((g1,g2) ∈ S) {
16   if ( (q; g1,g2) is a strict Buchberger triple )
17     S = S \ {(g1,g2};
18 }
19 S = S ∪ B;
20 foreach (p ∈ G) {
21   if ( lm(q) | lm(p))
22     G = G \ {p};
23 }
24 G = G ∪ q;
```

Algorithm 6: `Buchberger::updatePairs`: Filling the set of critical pairs with critical S-pairs.

this order, we can eliminate pairs by the second criterion, using pairs which are disjoint. As a third step, we eliminate new, strict Buchberger triples from our set of S-pairs. In the last step, we remove polynomials from G which are superfluous since their leading monomial is a multiple of the new elements leading monomial.

4.2.4 Strategies for reduction

Reducing polynomials with respect to a set of other polynomials is a key procedure when calculating a Gröbner basis. This procedure can be optimised in two directions.

One direction for improvement is the time used to find a suitable reductor for a polynomial p. During the Buchberger algorithm, the number of different polynomials often grows large. Iterating through all potential reductors and checking whether its leading monomial divides the leading monomial of p may take a lot of time. If the polynomials are arranged in order of increasing leading monomials, with respect to the used monomial order, as proposed in [CLO97, p. 108], then we might save some time for degree-based orders since smaller leading monomials are then more likely to divide another term. Moreover, the number of potential reductors is smaller, according to the following lemma.

Lemma 4.2.5. *Given polynomials $p, p' \in K[\bar{x}]$ with $0 \neq p \neq p'$ and a set of*

polynomials $P \subset K[\bar{x}]$. We define a subset of polynomials $Q \subseteq P$ such that the leading monomials of polynomials in Q are exactly those polynomials in P which are smaller than $\mathsf{lm}(p)$ w.r.t. the monomial order we use during the reduction, i.e., $Q = \{q \in P : \mathsf{lm}(q) \leq \mathsf{lm}(p)\}$. If $p \xrightarrow{P} p'$ then also $p \xrightarrow{Q} p'$.

Proof. From [BWK93, Theorem 5.5] we know that $\mathsf{lm}(q) \mid \mathsf{lm}(p)$ implies $\mathsf{lm}(q) \leq \mathsf{lm}(p)$. □

This brings a speed-up for the cases where there is no suitable reductor at all. This ordering however has its drawbacks. The larger the factor with which the reductor is multiplied the smaller the leading monomial of the reductor. This means that we might need more reduction steps, and that the polynomial which has to be reduced grows more with respect to the number of terms.

Based on the ideas in [PdMJ10] we can build an index which can further reduce the number of potential reductors. This can be done in two ways, but both of the index-strategies are based on the following observation:

Corollary 4.2.6. *Given polynomials $p, p' \in K[\bar{x}]$ with $0 \neq p \neq p'$ and a set of polynomials $P \subset K[\bar{x}]$.*

1. *We define the set of polynomials Q, such that in the leading monomials of polynomials in Q only variables occur which also occur in $\mathsf{lm}(p)$. $Q = \{q \in P : \forall x \in \mathsf{lm}(p).x \in \mathsf{lm}(q)\}$. Then it holds that $p \xrightarrow{P} p'$ implies $p \xrightarrow{Q} p'$.*

2. *We define the set of polynomials Q', such that in the leading monomials of polynomials in Q' at least one variable occurs which also occurs in $\mathsf{lm}(p)$, i.e., $Q' = \{q \in P : \exists x \in \mathsf{lm}(p).x \in \mathsf{lm}(q)\}$. Then it holds that $p \xrightarrow{P} p'$ implies $p \xrightarrow{Q'} p'$.*

Proof. 1. Assume $p \xrightarrow{P \setminus Q} p'$. Then $\mathsf{lm}(q) \mid \mathsf{lm}(p)$ for some $q \in P \setminus Q$. Let $\mathsf{lm}(q) = \bar{x}^{\alpha}$ and $\mathsf{lm}(p) = \bar{x}^{\beta}$. It follows from the definition that $\alpha_i \geq \beta_i$ for all $1 \leq i \leq |\bar{x}|$, and especially that $\beta_i > 0$ implies $\alpha_i > 0$. Now for an arbitrary $q \in P \setminus Q$ we know that there exists a $x \in \mathsf{lm}(p)$ with $x \notin \mathsf{lm}(q)$. Let j be the index of this x. Then $\beta_j > 0$, but $\alpha_j = 0$. Contradiction.

2. Clear from the fact that $Q \subset Q'$.

□

Based this observation we propose the following. Instead of searching for a suitable reductor in a long list of polynomials, we introduce a list l_x for each variable x. We have two possibilities to fill these lists.

1. The list l_x is filled with all polynomials p with $x \in \mathsf{lm}(p)$. If we now want to check whether a polynomial p can be (top-)reduced, we only have to check one of the lists l_x where $x \in \mathsf{lm}(p)$.

2. For each polynomial p we fill one of the lists l_x with $x \in \mathsf{lm}(p)$ with p. Now if we want to check whether a polynomial p can be (top-)reduced, we have to check all lists l_x where $x \in \mathsf{lm}(p)$.

The second direction in which improvements can be made is by trying to reduce the number of terms which appear during the reduction. A simple approach is by ordering polynomials in either the list or index explained above according to the number of terms. A more involved approach to reach this goal is called Slimgb and was introduced by Brickenstein in [Bri10]. It uses a cost function which may use the length as well as the degree and the coefficient to select suitable reductors. Furthermore, it calculates reductions in parallel and replaces some selections during the reduction.

4.3 State-of-the-art: Signature-based and saturation algorithms

Although the Buchberger algorithm enhanced with the aforementioned criteria and strategies performs well, recent development has lead to overall faster algorithms. We do not give details, but only make some short remarks.

Signature-based algorithms The main idea for this class of algorithms is that the cancellations are an algebraic object. The signature-based algorithms calculate a basis for these cancellations, such that the number of reductions is further reduced. However, calculating the basis for the cancellations may impose a large overhead. Initiated by the F5 algorithm by Faugère [Fau02], a lot of research is done to improve the practicality of the ideas from [MMT92]. Recent algorithms are given in e.g. [EP11] and [GGV10]. The structure of the algorithm closely resembles the Buchberger algorithm, as well as the necessary data structures.

Saturation algorithms The saturation algorithms, based on ideas from automatic theorem proving, are proposed by Passmore et al. in [PdMJ10]. These algorithms aim at so called large, largely linear input sets, as they appear in several SMT benchmarks. The algorithm is based on the abstract Gröbner bases theory from [PdM09], which allows one to show correctness of algorithms which do not consider all S-pairs. The algorithms keep the Gröbner basis small, and provide several optimised data structures for fast access. Since these algorithms are not obliged to check all S-pairs, a quadratic overhead for checking all combinations of S-pairs can be avoided. The main steps from the algorithm are however very similar to the Buchberger algorithm, and therefore, the same data structures can be used, as well as our approach for small conflict sets.

4.4 Efficient data structures

There are a couple of operations in the Buchberger algorithm which are performed numerous times. Previously, we mostly discussed algorithmic ways to reduce the number of these operations. In this section, we go to a technical level and discuss the data structures used in our algorithms.

4.4.1 Monomials and polynomials

Let us first discuss the basic objects, the polynomials. In general, multivariate polynomials p are sparse, i.e., for a polynomial p only a small fraction of all monomials $m < \mathsf{lm}(p)$ appear with a non-zero coefficient in p. For the Gröbner basis algorithm, adding monomials and getting the leading monomial are important operations, but we want a generic data structure which can also be used efficiently in other operations. Therefore we decided to use a dynamic array of monomials with non-zero coefficient for the polynomials.

Especially in the context of SMT, the number of variables in the input is usually large. Moreover, since we work incrementally and additional variables might be introduced during the calculation, we do not have a fixed bound on the number of variables. In the monomials, often only a small number of variables occur. We store monomials as dynamic arrays of pairs, which match a variable with the degree in which it occurs in the monomial. The data structure for monomials furthermore saves a rational coefficient, and since we request the total degree regularly, we also store this. We depicted an example in Figure 4.4.

In [BS98] a different monomial representation is used: Fixed size arrays of exponents are used, in which the position of the exponent yields the corresponding variable. With this representation *packing* is possible, i.e., letting one machine word represent multiple exponents. Advantages of this are that less memory is consumed (better cache performance) and less operations are necessary for division and multiplication. Obviously, the maximal degree of each variable gets smaller the more exponents are packed into a single machine word. Therefore, only a limited number of exponents are packed in a single machine word. Since the position of an exponent is used to identify its corresponding variable, the number of machine words used is given by the number of (potential) variables divided by the number of exponents packed into a machine word. Whereas in the storage used in GiNaCRA the number of machine words needed for each monomial is given by the number of variables occuring in the monomial times two.

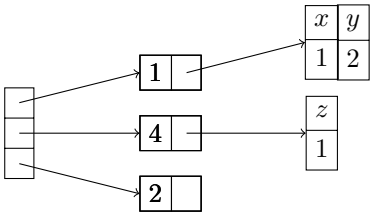


Figure 4.4: Example storage of the polynomial $xy^2 + 4z + 2$.

4.4.2 Finding suitable reductors

For the generated ideal we provide a class, which consists of a dynamic array in which the input polynomials as well as the reduced S-polynomials are saved. In this way, an ideal-object can provide static indices, which are used in, for example, the S-pairs. For finding suitable reductors, this dynamic array would lack performance, so the object additionally stores a pointer to another data structure which has references on the polynomials in the ideal. In this structure,

the potential reductors can be stored and sorted in an arbitrary fashion, as any access and updates are performed through the ideal object.

For fast look-up of reductors, the strategies from Section 4.2.4 can be implemented. Other refinements, given in [RS] include:

- Division-masks, such that only a single machine operation is necessary to exclude a reductor.
- KD-trees, such that a more efficient search among the reductors can be implemented.

The reductors are currently ordered according to either the number of terms or the monomial order.

4.4.3 Reduction

Even with the criteria discussed in Section 4.2.1 the reduction procedure is the most costly within the algorithm. We provide a small data structure dedicated to storing the polynomial during reduction. The reduction frequently has to

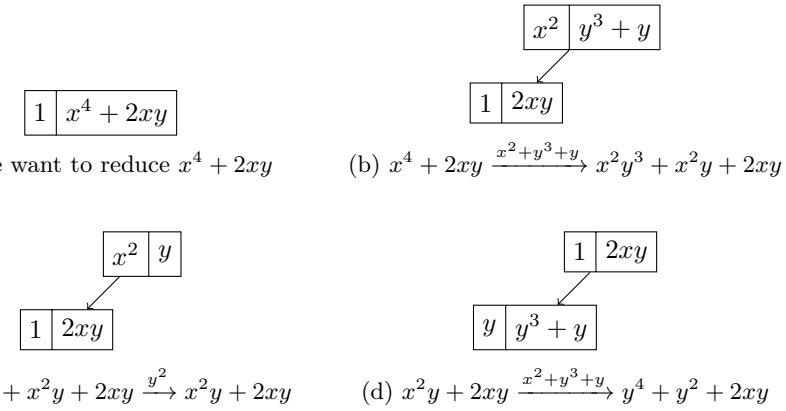
- pop the leading monomial, and
- merge the current polynomial p with a monomial m times a polynomial q.

Directly merging monomials into an array of monomials might lead to a very bad worst-case behaviour, as shown by Yan in [Yan98]. To overcome this, geo-buckets were introduced in the same paper. Another way of saving terms is in heaps, as proposed by Johnson in [Joh74]. Geo-buckets perform less comparisons, but hashes seem to have a better overall performance on multiplication and division due to cache effects [MP07]. In [MP11], Monogan and Pearce present more involved algorithms for division with a heap. Roune and Stillman propose the use of a third priority queue called tournament trees in [RS]. All priority queues can be further enhanced to reduce the number of comparisons needed during merging. We give a short overview.

From Johnson [Joh74] originated the idea of *compression*. Instead of having monomials in a node of the priority queue, a monomial factor and a polynomial is saved. The heap is then sorted according to the leading terms times the monomial factors in each node. The observation why this works is simple: polynomials are already ordered when they are merged into the current polynomial.

Another approach to reduce the number of comparisons is *active deduplication* from [RS]. During insertion in the tree, equal monomials are merged by summing up their coefficients. Based on this idea we introduce *passive deduplic-tion*, which eliminates duplicates while popping the leading term. Notice that the passive variant only prevents unnecessarily many searches for reductors.

The idea of hashing monomials to find equal monomials is mentioned by Fateman in [Fat03], but results were not quite promising. The implementation based on LISP seems to suffer from the effects of garbage collection. In the context of reduction, and within a C++ implementation, hashing performs well [RS]. Notice that hashing alone does not suffice, because hashes do not provide any ordering.



(a) We want to reduce $x^4 + 2xy$

(b) $x^4 + 2xy \xrightarrow{x^2+y^3+y} x^2y^3 + x^2y + 2xy$

(c) $x^2y^3 + x^2y + 2xy \xrightarrow{y^2} x^2y + 2xy$

(d) $x^2y + 2xy \xrightarrow{x^2+y^3+y} y^4 + y^2 + 2xy$

Figure 4.5: Example with the reductor data structure (heap and compression).

In comparison, [RS] showed experimentally that hashing perform well under arbitrary instances, while the performance of active deduplication and compression depend on the instance.

In GiNaCRA, we provide a reductor class which can work with any priority queue. Currently, we work with a compressed representation with passive deduplication on a heap. An example is shown in Figure 4.5.

4.4.4 S-pairs

In Section 4.2.2 we discussed strategies to select the next S-pair. An important observation from [RS] is that S-pairs are often added in bunches, and that sorting them is usually easy. We can then add the sorted list to a heap of such lists. This way, we have less comparisons while keeping the S-pairs sorted according to our strategy. More involved storage schemes are also given in [RS], but are not subject of our research.

Chapter 5

Applying the Real Nullstellensatz

From Section 3.3.2 we can conclude that given a sum of squares p and an ideal I, if $(p+1) \in I \subset \mathbb{Q}[\bar{x}]$ then $V_{\mathbb{R}}(I) = \emptyset$. In this chapter, we discuss how this criterion can be used to develop a module which can show inconsistency over the reals of a set of equalities. The method is closely resembling the approach presented in [PQR09] and implemented in KeYmaera [PQ08].

5.1 Finding witnesses by sums of squares

Given a set of polynomials I, checking whether for a given polynomial p, e.g., a sum of squares, it holds that $p \in \langle I \rangle$ is easily done with the help of a Gröbner basis for I. Thus, in order to develop an efficient method which tries to find any sum of squares p with $p \in \langle I \rangle$, we need to provide an efficient procedure for finding a polynomial which makes sure that the polynomial is a sum of squares.

As a first idea, we could use a brute-force approach, iterating over all polynomials and checking if it is a sum of squares, which is discussed later on, and if $\mathsf{red}_G(p+1) = 0$ holds. For all but very small problems, this approach does not seem to be very promising. Though this is the basic idea of our approach. instead of iterating over polynomials, we will iterate over monomials and append them in a vector of monomials, which yields a more targeted search.

As a first step, we notice that we can represent matrices by vector-matrix-vector products.

Proposition 5.1.1 (Matrix representation)**.** *[Par03, Section 3.2] Let $p \in \mathbb{Q}[\bar{x}]$ with $\mathsf{tdeg}(p) \leq 2d$ for some $d \in \mathbb{N}$, and z a vector of all monomials $m \in M_{\bar{x}}$ with $\mathsf{tdeg}(m) \leq d$. Then it holds that $p = z^t Q z$ for some suitable matrix Q. A polynomial written as such a vector-matrix-vector product is called* matrix representation*. The set $L = \{Q : z^t Q z = p\}$ of all such matrices is an affine subspace.*

Let in the following denote L_p the set of all matrices $\{Q : z^t Q z = p\}$.

Since most of the multivariate polynomials which occur in our computations are sparse, we often want a smaller vector z of monomials.

Definition 5.1.1 (Reduced matrix representation)**.** *Let the polynomial $p = z^tQz$ be in matrix representation. Let J be the index set of empty rows and columns, defined as $J = \{i : \text{ for all } 1 \leq j \leq \dim(z)\ Q_{i,j} = 0 \wedge Q_{j,i} = 0\}$. Then $\tilde{z}^t\tilde{Q}\tilde{z}$, with $\tilde{z}$ obtained from z by removing all entries z_i with $i \in J$ from z, and $\tilde{Q}$ obtained from Q by removing all entries $q_{i,j}$ with $i \in J$ or $j \in J$ from Q, is called the* reduced matrix representation*.*

Example 5.1.1. *We write a polynomial in matrix representation and in reduced matrix representation.*

$$\begin{aligned} & 2 + y + 3y^2 \\ &= \begin{pmatrix} 1 & x & y \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \\ y \end{pmatrix} \\ &= \begin{pmatrix} 1 & x \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ y \end{pmatrix} \end{aligned}$$

Now, instead of iterating over all polynomials, we can iterate over vectors of monomials and matrices.

By the following theorem, we are able to consider only polynomials which are a sum of squares.

Definition 5.1.2. *A matrix $M \in \mathbb{R}^{k\times k}$ is* positive semidefinite *(PSD) if M is symmetric ($M = M^t$) and $x^tMx \geq 0$ for all $x \in \mathbb{R}^k$.*

Theorem 5.1.2. *[Par03, Theorem 3.3] Let $f \in K[\bar{x}]$ with $\mathsf{tdeg}(f) \leq 2d$. Then the following are equivalent:*

1. *f is a sum of squares.*
2. *For a vector z which contains all monomials $m \in M_{\bar{x}}$ with $\mathsf{tdeg}(m) \leq d$ there exists a matrix Q with $f = z^tQz$ s.t. Q is PSD.*
3. *There exists a vector of monomials z and a matrix Q with $f = z^tQz$ s.t. Q is PSD.*

By this theorem, instead of iterating over all polynomials we can iterate over all vectors of monomials and PSD matrices and we then only have to check if for the represented polynomial p the equation $\mathsf{red}_G(p + 1) = 0$ holds.

5.1.1 Using semidefinite programming

Before we proceed refining our search for sum of squares, we show how to check whether a given polynomial is a sum of squares. This method is not directly used in our method since we search within the PSD matrices for a suitable matrix representation directly, but the idea is very similar.

Given a polynomial p and the set L_p, we want to check whether p is a sum of squares. With Theorem 5.1.2 we can follow that p is a sum of squares if and only if the intersection of L_p with the set of all PSD matrices is non-empty. This can be efficiently done by using *semidefinite programming* (SDP), which we introduce next.

Definition 5.1.3 (Trace of a matrix)**.** *Let $A \in \mathbb{R}^{n\times n}$ be a matrix. The* trace *of A, $\mathsf{trace}(A)$, is the sum of the diagonal elements $\sum_{i=1}^{n} a_{i,i}$.*

It follows directly that $\mathsf{trace}(A,B)$ is equivalent to the sum $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}b_{j,i}$.

Definition 5.1.4 (Semidefinite programming)**.** *Semidefinite programming is an optimisation problem:*

$$\begin{array}{lll} \textit{Maximise} & \mathsf{trace}(C,X), & C \textit{ symmetric} \\ \textit{subject to} & \mathsf{trace}(A_i,X) = b_i, & A_i \textit{ symmetric},\ 0 \le i \le m \\ & X \textit{ PSD.} & \end{array}$$

A detailed description of SDP, and methods to solve this problem can be found in, e.g., [VB94]. Next we give a very brief description of the intersection of L_p with the set of PSD matrices, based the detailed treatment in [Par03, Section 3].

In order to decide whether a polynomial p is a sum of squares, we write it in matrix representation. By matching coefficients, we get linear constraints describing L_p.

Example 5.1.2. *Let us reconsider the polynomial $3y^2 + y + 2$ from Example 5.1.1. By comparing coefficients, we get $q_{1,1} = 2, q_{1,3} = 1$, and $q_{3,3} = 3$. By SDP, we find that no matrix subject to these constraints is PSD.*

We come back to the refinement of our search. We want to find a sum of squares s s.t. $\mathsf{red}_G(s+1) = 0$. In the next part, we reformulate the search. Afterwards, we show that we can encode $\mathsf{red}_G(s) = 0$ as constraints for the SDP, as first shown in [PQR09]. We conclude extending this to $\mathsf{red}_G(s+1) = 0$.

Reformulation of the problem First, we notice that by expanding the matrix representation z^tQz, we get a linear polynomial with respect to the entries in Q. Furthermore, we get all products of two monomials in z.

Example 5.1.3. *We take a symbolic example showing the reformulation.*

$$\begin{aligned} & \begin{pmatrix} z_1 & z_2 & z_3 \end{pmatrix} \cdot \begin{pmatrix} q_1 & q_2 & q_3 \\ q_4 & q_5 & q_6 \\ q_7 & q_8 & q_9 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \\ = {} & z_1z_1 \cdot q_1 + z_1z_2 \cdot q_2 + z_1z_3 \cdot q_3 + \\ & z_1z_2 \cdot q_4 + z_2z_2 \cdot q_5 + z_2z_3 \cdot q_6 + \\ & z_1z_3 \cdot q_7 + z_2z_3 \cdot q_8 + z_3z_3 \cdot q_9 \\ = {} & \begin{pmatrix} q_1 & q_2 & \cdots & q_8 & q_9 \end{pmatrix} \cdot I \cdot \begin{pmatrix} z_1z_1 \\ z_1z_2 \\ \cdots \\ z_2z_3 \\ z_3z_3 \end{pmatrix} \end{aligned}$$

with I the identity-matrix.

Thus, we can rewrite $p = z^tQz$ to q^tCm with vectors q of coefficients, and m of monomials, both of dimension $\dim(z)^2$.

Now, since multiplication is commutative, we can join monomials $z_i z_j$ and $z_j z_i$. Because Q is symmetric, we replace all q_{ji} by q_{ij} with $i < j$. We have to duplicate those entries in the coefficient matrix which are multiplied by a $z_i z_j$ with $i \neq j$.

Now, with this representation, we can extract the constraints for the SDP.

Extracting constraints for the SDP As a first step, we calculate the remainder with respect to our Gröbner basis G. First we give a definition of reducing polynomials with coefficients.

Definition 5.1.5. *The* remainder *of a polynomial p over the variables $\bar{x}$ and coefficients $\bar{q}$ w.r.t. a Gröbner basis over $\bar{x}$ is defined as follows:*

$$\mathsf{red}_G(\sum_{i=1}^{n} m_i p_i) = \sum_{i=1}^{n} m_i(\mathsf{red}_G(p_i))$$

with $p_i \in K[\bar{x}]$ and $m_i \in M_{\bar{q}}$ for all $1 \leq i \leq n$.

We set $q^t D m' = \mathsf{red}_G(q^t C m)$, in which m' contains all monomials which appear in $\mathsf{red}_G(m_i)$ for some $1 \leq i \leq \dim(m)$. Since the monomials are all smaller w.r.t. the monomial order of G, $\dim(f')$ is finite. D is a suitable matrix, which exists by the same reasoning we applied to the existence of the matrix C. Notice that all monomials in m' are linear independent. Thus $q^t D m' = 0$ if and only if $q^t D = 0$. This yields a linear equation for each monomial in m'.

Example 5.1.4. *[PQR09] Given the Gröbner basis $G = \{a^2 - x + y,\ b^2 - z,\ c^2xz - c^2yz + 1\}$, and the vector of monomials $z = (1,\ a,\ abc)^t$. We find a sum of squares s with $\mathsf{red}_G(s) = 0$, by applying SDP with the following constraints.*

From expanding the matrix representation we get:

$$\mathsf{red}_G(s) = \mathsf{red}_G(q_{1,1}1^2 + 2q_{1,2}a + 2q_{1,3}abc + q_{2,2}a^2 + 2q_{2,3}a^3bc + q_{3,3}a^2b^2c^2)$$

Now we extract the matrix coefficients as parameters:

$$\begin{aligned} 0 =& \mathsf{red}_G(q_{1,1}1^2 + 2q_{1,2}a + 2q_{1,3}abc + q_{2,2}a^2 + 2q_{2,3}a^3bc + q_{3,3}a^2b^2c^2) \\ =& q_{1,1}\mathsf{red}_G(1) + q_{1,2}\mathsf{red}_G(2a) + q_{1,3}\mathsf{red}_G(abc) + \\ & q_{2,2}\mathsf{red}_G(a^2) + q_{2,3}\mathsf{red}_G(2a^3bc) + q_{3,3}\mathsf{red}_G(a^2b^2c^2) \\ =& q_{1,1} - q_{1,3} + q_{1,2}2a + q_{1,3}2abc + q_{2,2}x - q_{2,2}y + q_{2,3}2abcx - q_{2,3}2abcy \end{aligned}$$

Since all different monomials are linearly independent, we get the following set of equations

$$\begin{aligned} (1):&\quad q_{1,1} - q_{1,3} = 0 \\ (a):&\quad 2q_{1,2} = 0 \\ (abc):&\quad 2q_{1,3} = 0 \\ (x):&\quad q_{2,2} = 0 \\ (y):&\quad -q_{2,2} = 0 \\ (abcx):&\quad 2q_{2,3} = 0 \\ (abcy):&\quad -2q_{2,3} = 0 \end{aligned}$$

This notion is formally captured by the following lemma.

Lemma 5.1.3. *[PQR09, lemma 2] Given a Gröbner basis $G \subset \mathbb{Q}[\bar{x}]$. Assume $p = q^tCm$ and $p' = q^tDm$ are two matrix representations $p, p' \in \mathbb{Q}[\bar{q}][\bar{x}]$ s.t. $p' = \mathsf{red}_G(p)$. Let the entries in m be pairwise different. Then the following holds:*

$$\{x \in \mathbb{R}^k : \mathsf{red}_G(x^tCm) = 0\} = \{x \in \mathbb{R}^k : x^tD = 0\}.$$

This can be equivalently formulated as constraints in the SDP problem.

We return to the original problem of $\mathsf{red}_G(1+q^tCm) = 0$. Since the 1 in this sum is the only term which has no coefficient parameter q, and $\mathsf{red}_G(1) = 1$ for all Gröbner bases, we get $\mathsf{red}_G(q^tCm) + 1 = 0$. The 1 is a constant monomial, and to capture it in our SDP-constraints, we introduce the convention that $m_1 = 1$. Then there is always a constraint for the constant monomial. By convention, this will always be the first constraint. By adding -1 on both sides of the first equation we get the following SDP formulation:

$$\begin{array}{rll} \text{Maximise} & \mathsf{trace}(C,X), & C \text{ symmetric}, \\ \text{subject to} & \mathsf{trace}(A_1,X) = -1, & A_1 \text{ symmetric}, \\ & \mathsf{trace}(A_i,X) = 0, & A_i \text{ symmetric}, 2 \leq i \leq m, \\ & X \text{ PSD}. & \end{array}$$

5.1.2 Gaining an exact solution

Using exact algorithms for SDP makes the problem highly intractable, as has been indicated by [PP08]. Furthermore, although there are many free libraries for SDP, e.g. CSDP [Bor99] and SDPA [FFK+08], they all use numerical methods. Thus, for a given SDP and a solution matrix over floating-point numbers, we ought to find a solution matrix over the rational numbers. We do this in three steps, which are discussed in greater detail in this section.

First, we approximate the solution-matrix entries by rationals with small nominators and denominators. Then, we modify these entries such that the rational solution matrix satisfies the constraints of the SDP. The last step is then a certification that this rational solution is indeed a PSD matrix.

Finding small rational approximations for floating points. Since using the conversion of a float to a rational number which returns a rational with a very high precision tends to yield very big nominators and denominators, we want to find a sufficient precise, rational number with nominator and denominator. We use the Stern-Brocot tree [GKP94] (see Figure 5.1) with separate nominators and denominators for all entries. Harrison [Har07] pointed out that he experienced better results with more uniform denominators, but these are certainly bigger if working with the same precision. This increases the time used for the following steps. The idea behind the transformation is to traverse the tree until we found a solution which is within a certain error. During the traversal, we only expand the the nodes we visit.

Example 5.1.5. *Given the floating point 0.62, we look for a rational number with a maximal error of 0.025. We start at the root of the tree, and compare the float with the value in the root (1). Since the float is smaller, we expand*

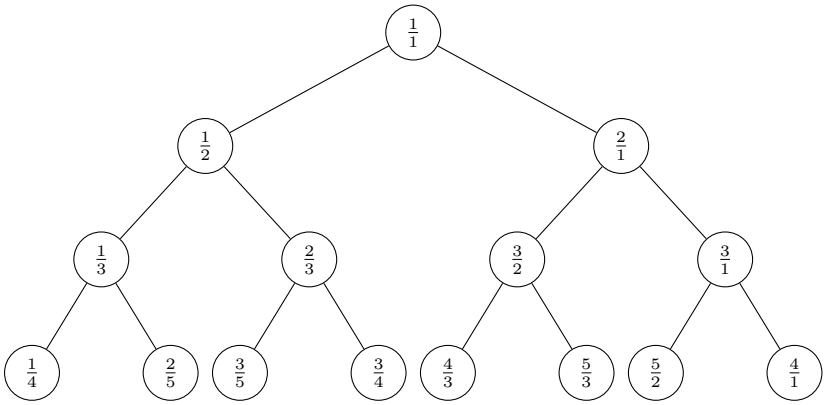


Figure 5.1: The topmost part of the Stern-Brocot tree.

the left child. We again compare the float with the value, which is $\frac{1}{2}$*, thus we expand the right hand side, compare again, expand the left hand side, and have the value* $\frac{3}{5} = 0.6$*, which is within the error margin.*

Finding a rational solution for a linear equation system. The entries x_i of our solution matrix are subject to the constraints A_i from the SDP, whose entries are gathered in the rows a_i. Moreover, our solution should be symmetric, so we add rows b_i which ensure that the solution matrix is symmetric. Thereby we get a linear equation system (with the right hand side equal to the SDP formulation).

$$\begin{pmatrix} --- & a_1 & --- \\ & \vdots & \\ --- & a_m & --- \\ --- & b_1 & --- \\ & \vdots & \\ --- & b_t & --- \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

To solve this, we use a slightly modified Gauss-algorithm, in which after the matrix is brought into row-echelon form, we proceed as usual, with the modification that if a row in the matrix consists of zeroes only, then we do not take an arbitrary entry as solution, but the solution from the approximate rational solution. In order to increase the probability that this solution is near to our floating point solution, we start by permuting the columns of the equation system such that that our approximate solution-entries with higher precision are more likely to be taken.

Certifying that the matrix is PSD A trivial way of certifying that a matrix is PSD follows directly from the following lemma:

Lemma 5.1.4. *[GVL96, Chapter 4] A matrix A is PSD if a Cholesky factorisation* $A = LDL^t$ *exists.*

We thus attempt to compute the Cholesky factorisation. If we succeed, then our rational matrix is indeed a solution and we can easily extract our witness

s with $\mathsf{red}_G(s+1) = 0$. If we did not succeed, we might try the routine again with a higher precision for the transformation of the floating point numbers into rational numbers.

5.2 A module based on the real Nullstellensatz

We have implemented the calls to our real Nullstellensatz based procedure within the module from Section 4.1. We show the main algorithm and then shortly discuss SMT-compliance.

5.2.1 A general algorithm

After we calculated a Gröbner basis and if this Gröbner basis is not equal to $\{1\}$, we make a call to Algorithm 7. We iteratively add monomials to our SDP problem, until either the SDP procedure finds a PSD matrix and returns `true`, or the upper bound on the monomials is reached. From the monomials and

```
1  Input: Gröbner basis G, maximal degree d.
2  Output: Witness for infeasibility if unsat, else 0.
3  M = {m ∈ M_x̄ : tdeg(m) < d};
4  N = []; // empty list of monomials.
5  C = {C_m ∈ Q^(|M|×|M|) : m ∈ M}; // constraint matrices.
6  while( M ≠ ∅ ) {
7    select m ∈ M;
8    M = M \ {m};
9    N.append(m);
10   foreach( n_i ∈ N ) {
11     t = red_G(n_i · m);
12     while ( t ≠ 0 ) {
13       C_lm(t)(i, |N|) = lc(t);
14       t = t − lmc(t);
15     }
16   }
17   (r, S) = SDPInterface(C);
18   if( r == SDP success ) {
19     w = GainExactSolution(S, C);
20     return w;
21   }
22 }
23 return 0;
```

Algorithm 7: Core procedure to produce SDP.

the Gröbner basis we have to form the constraints. We then hand over the constraints to the SDP procedure.

As pointed out in the last section, we have to gain an exact solution from the floating point solution the SDP may return. We do this according the description in Section 5.1.2.

We now discuss the key components in greater detail.

Monomial iterator The monomial iterator generates the monomials which are used as entries in our monomial vector.

Obviously, we only have to generate monomials over variables which appear in the Gröbner basis. But even some variables from the Gröbner basis can be skipped. For example, for identities appearing in the Gröbner basis such as $x - y$, x is substituted by y in all but this equality during the Gröbner basis computations. Since the identity yields that the assignments to x and to y are equal within the variety, we can leave out the variable x while iterating over the monomials. More complex ways to leave out other variables during the generation of monomials are given in [PQR09].

The current monomial iterator generates a bunch of monomials at once, and then adds them one by one to our core procedure from Algorithm 7. The monomials are created according to ascending graded lexicographic order. A change to the order has a major impact on the performance of the method, as finding monomials which generate a witness is the key concept of the procedure.

SDP procedure As backend for the SDP problems we use CSDP [Bor99], which was also used by [Har07] and [PQR09]. Before CSDP is called, we transform the data structures to match the precise call. Notice that we cannot simply extend the data structures from a previous call, which yields some overhead.

5.2.2 A note on SMT-compliance

We currently only support small conflict set generation. The witness which is returned by our procedure is by construction in the ideal generated by the Gröbner basis. If we reduce the witness with respect to the Gröbner basis, the remainder is zero. The witness can be generated by the polynomials that are used during the reduction. This smaller set is thus already inconsistent. Our reduction procedure appends a reason vector from which Gröbner basis elements that were used during reduction are easily extracted. Regarding incrementally, we can leave out monomials that reduce to zero. This decreases the time that we need for iterating over the monomials, but does not affect the SDP procedure, which is currently clearly the most time consuming part of the procedure.

Chapter 6

Experimental results

In this section, we give the running times of some experimental runs of the SMT solver using our module. We then give our interpretation based on these results.

In order to attain these results and get a better overview of the effects of single components within our module, we composed several binaries based on the release build of GiNaCRA 0.7.0 and SMT-RAT 0.3.0[1] and compared them with the base SMT-RAT solver as well as with Z3 4.0.

The base SMT-RAT solver consists of a VS module with the CAD module as backend module. Z3 4.0[2] is the successor of the winner of last year's SMT Competition in the category QF_NRA (Quantifier Free Nonlinear Real Arithmetic).

We applied the Gröbner basis module by using the following strategy (B): The base SMT-RAT solver is extended by the Gröbner module as simplification module, which uses the base strategy (VS, CAD) as a backend. The order in which a theory call is processed is thus GB, VS, CAD.

This and other strategies which involve a Gröbner basis are tested with some variations of the settings. All Gröbner bases are calculated with respect to the graded lexicographic order. The following combinations of settings appear in the results:

oc The Gröbner basis is calculated and only used to check consistency.

p The Gröbner basis is calculated and passed instead of the received equalities.

i As *p*, however inequalities are reduced modulo the Gröbner basis. The passed inequalities are the received inequalities which do not reduce to `true`. If a conflict is found, `unsat` is returned directly.

ri As *i*, but the reduced inequalities are passed instead.

pri As *ri*, but in case of a conflict, we proceed and handle the other inequalities as well to detect multiple conflicts.

at Here, all inequalities are transformed into equalities. The Gröbner basis is calculated, but not passed.

[1] Notice that these versions also include several improvements which are not in the scope of this thesis. As a result, this section cannot be compared with the experimental results given in [CLJÁ12].

[2] Binary available from `http://rise4fun.com/z3/`.

Settings marked with a subscript "+" denote those which extract smaller reasons from the reason vectors, whereas those without use the naive over-approximation of the conflict set.

The settings of the other modules are fixed. To improve the results and let the solver benefit more from our module, it would be advisable to tweak the settings of other modules in order to reflect the behaviour of our module.

All benchmarks were run on systems running Debian "squeeze", with a 2x 2.33 GHz. Intel® Xeon® processor with 4 MB cache. The maximal memory consumption was limited by software to 4 GB, but this limit was never reached. The results are given in Table 6.1.

	Bouncing ball			Rectangular pos.			Keymaera		
	80			22			421		
	sol.	acc.	cmp.	sol.	acc.	cmp.	sol.	acc.	cmp.
base	31	481.0	14.9	20	569.4	56.4	413	1931.4	200.5
$B(oc)$	31	548.3	16.6	20	427.7	33.6	413	1847.5	196.6
$B(oc)_+$	28	244.7	17.5	21	938.6	25.3	413	1768.2	195.8
$B(p)$	25	93.7	25.9	19	316.4	36.3	411	647.2	477.8
$B(p)_+$	24	31.3	31.3	20	503.9	35.9	411	648.6	477.0
$B(i)_+$	24	31.4	31.4	16	54.8	34.8	410	587.6	408.9
$B(ri)_+$	25	43.5	29.3	17	56.7	36.2	412	620.4	431.7
$B(pri)$	24	32.3	32.3	15	60.5	51.6	411	597.5	408.1
$B(pri)_+$	25	43.0	28.4	16	71.5	54.3	411	598.4	410.5
$B(at)_+$	30	251.4	16.9	16	588.5	278.0	415	1380.5	1190.7
Z3	80	5.5	1.1	22	45.2	1.5	419	4.2	3.7

Table 6.1: Running times in seconds of several binaries on three benchmark sets.

The structure of the table is as follows. On top, the name of the benchmark set and the number of problem instances contained in it are given. Then, for each combination of binary and benchmark, there are three entries. The *solved instances* (sol.) gives the number of instances which were solved within the time limit of 600 seconds. The *accumulated time* (acc.) is the total time the binary took on the solved instances. The *compare time* (cmp.) is the total time the binary spent on solving those instances which were solved by all the solvers. Detailed results are available upon request.

The selected sets are motivated by a practical context. The benchmarks in Bouncing ball are from a model of the nonlinear movement of a bouncing ball. Rectangular positioning is a set of benchmarks which describe placements of small rectangles in a larger rectangle. These benchmarks were taken from [CLJÁ12]. The Keymaera benchmark is a set of benchmarks from the hybrid systems verification tool KeYmaera. These benchmarks were taken from [JdM12].

We see directly that Z3 has the best performance. Z3 seems to benefit a lot from three modules which integrate methods[1] not yet available in SMT-RAT. These are an integrated simplex solver for the linear fragment, as well as interval constraint propagation and a SAT-style CAD with bounded intervals [JdM12].

[1] Z3 is closed source, and although a lot of options are available, we could not extract any information which method is the main reason for the better performance.

Moreover, SMT-RAT is a very young project in which there is still a lot of room for optimisation[1]. The SMT-RAT solver is meant to be a proof of concept that from several modules, an efficient solver for NRA can be built, therefore, we focus on the comparison between the different SMT-RAT solvers.

A first observation is that on many instances, there is a significant speed-up when we calculate the Gröbner basis, as some conflicts are found faster then when using virtual substitution. However, if we pass the Gröbner basis, the solver performance drops. We thereby conclude that a Gröbner basis is, in general, not a simplification for the virtual substitution.

It is difficult from the examples to say something about incrementality and backtracking. However, the effects of the smaller conflict set are made visible. Returning smaller conflict sets is in general a good idea, however, on some instances, due to the decisions the SAT solver makes, we run into some difficult theory calls, which do not appear with the bigger conflict sets returned. The greater the difference between running times for a single theory call for a given input formulae, the bigger this effect can get. We are confident that with some improved methods and some more influence on the decisions the SAT solver makes, smaller conflict sets will mostly improve the solver's performance.

Although the original focus was on equalities, a lot of benchmarks have a majority of inequalities, so the handling of these is of great interest. Handling inequalities by an inequalities table however seems to worsen the performance. Despite the overall performance drain, on some Keymaera benchmark instances, the performance increases significantly. We think that the reason is that the inequalities table produces overhead if there are no conflict sets which consist of one inequality and a set of equalities. On most benchmarks, these cases are either rare, or the conflict sets are easily found by the VS module as they only need a small number of substitutions. In the future, with integration of theory deductions, the SMT solver will make significantly smaller theory calls, and the overhead might be reduced. Furthermore, more involved selection of the inequalities to be reduced and the moments to do so might reduce the overhead while retaining the gain of simplifying complex inequalities.

Because transforming inequalities into equalities introduces a large number of extra variables we thought it to be inapplicable. The running times indicate that this approach is slow, but promising, as it is able to solve quite a lot of the benchmarks. The transformation allows the module to also detect conflict sets which consist of more than one inequality. We have not yet implemented all proposed methods for input sets with many variables, but the running times indicate that this might be necessary to handle larger sets, e.g., on Bouncing ball times drop on instances with a large number of variables. Notice that passing the transformed inequalities is not a good idea, as the high number of variables makes the virtual substitution very slow[2].

The Gröbner module might have more impact if used otherwise. Based on some analysis with the tool Valgrind[3], we can see that especially theory calls which involve the CAD module take a lot of time. Therefore, we might try to

[1] On some instances, the performance of SMT-RAT raised sharply and unexpected, this might be due to some optimisation for special cases

[2] This information cannot be extracted from Table 6.1, but some separate experiments showed that with such settings, only 3 instances of Rectangular positioning can be solved.

[3] Valgrind is a tool for debugging and profiling and is freely available from `http://valgrind.org/`

reduce the number of cases in which the CAD module is called by using the GB module to preprocess the passed formulae from the VS module. The strategy (C) used for this is embedded in the following set of binaries, which are base SMT-RAT solvers in which the calls to the CAD solver are simplified by the Gröbner module. The order of the modules is thus VS, GB, CAD. Binaries with this configuration were created with several settings for the Gröbner module, in which the same identifiers are used. The running times are depicted in 6.2. Rectangular positioning is left out, because the VS module never calls its backends on these instances.

	Bouncing ball			Rectangular pos.			Keymaera		
	80			22			421		
	sol.	acc.	cmp.	sol.	acc.	cmp.	sol.	acc.	cmp.
base	31	481.0	14.9	20	569.4	56.4	413	1931.4	200.5
$C(oc)_+$	31	135.5	14.5				412	1753.6	196.0
$C(p)_+$	31	135.9	14.6				413	1514.8	197.3
$C(i)$	30	45.9	14.6				405	196.7	196.7
$C(pri)_+$	30	43.4	14.4				405	195.6	195.6
$C(at)_+$	31	266.4	14.6				413	1573.6	194.4

Table 6.2: Running times in seconds of several binaries.

Here, the harder bouncing ball instances are solved a lot faster and thus the VS backend calls have an improved performance. Passing the Gröbner basis does not really affect the speed of the CAD. We have also noticed that the passed formulae produced by the VS module seem quite complicated. It might be more efficient in some cases to use the original theory call instead.

Chapter 7

Conclusion

7.1 Summary

In this thesis, we took two methods, Gröbner bases computations and semidefinite programming, which are heavily used in several computational tasks, and embedded them into an SMT-framework. The main motivation for this comes from automatic theorem provers, in which it is beneficial to have several methods, each specialised on fragments of real algebra, available in order to solve the input problem efficiently.

The used methods are specialised on equalities, and monomials and polynomials are the key object they work with. Therefore, we started this thesis defining the notion of consistency in algebraic terms. Based on this notion, we were able to apply Gröbner basis computations to simplify sets of constraints, and, based on the Weak Nullstellensatz, we could also decide unsatisfiability over the complex numbers for sets of equalities. While realising incrementality is straightforward, backtracking and small conflict set generation are more involved. We applied naive methods which seem to work well in practice.

Although our focus was on equalities, in the currently available benchmarks most problem instances consist, for the most part, of inequalities. Thus, in order to speed up these instances, we discussed two different approaches. One is tailored towards simplification and reduces the polynomials modulo the Gröbner basis, and the other one, which is more tailored towards reducing the number of backend module calls, is based on transforming inequalities into equalities by using extra variables. The inequalities table introduces much overhead at the moment, and the number of calls which benefit from it is limited. Transforming inequalities slows down the solver, but is able to solve more instances.

One major drawback of the Gröbner basis method is that all complex, thus not only the real, zeroes are taken into account. We use an approach based on semidefinite programming to apply the Real Nullstellensatz in order to detect inconsistencies over the reals for a set of equalities. The method lacks speed and is currently not able to handle the high number of variables, but we are aware of some methods to improve this. Yet, the approach does not gain any information if the set of equalities is satisfiable, which is the case for most theory calls from our benchmark sets. The approach is based on numerical algorithms, and thus, numerical solutions have to be transformed into exact, rational solutions. This

component worked surprisingly well.

In the last part, we showed through some experimental results that even a preliminary, SMT-compliant Gröbner basis computation speeds up the theory solver on average, but that further enhancements such as handling inequalities or applying the Real Nullstellensatz do have to be applied based on some heuristic. Interesting is the increased performance on those instances in which conflicts between equalities occur, because these are handled much faster than with the already available methods. This shows the importance of developing even more methods tailored towards specific fragments of real algebra.

7.2 Discussion

The performance of the Gröbner basis module with fixed settings varies between different instances. For the development of heuristics which select the right settings, we need a more fine-grained knowledge on what happens and how successive modules perform on certain actions and instances. For example, replacing the Gröbner basis by a new one removes several constraints from the successors, which may impose a larger overhead than if we would just extend the call with new equalities. Moreover, we saw different settings to have the best performance depending on the strategy used in the experimental results. Therefore, the Gröbner module should always be regarded in context of the used strategy.

With an algebraic definition of consistency, one might wonder if there are other algorithms within the computational algebraic geometry which may be beneficial to SMT. Although a Gröbner basis is reduced in some sense, the polynomials may get a lot bigger, which makes Gröbner basis not suitable as simplification in general. Methods tailored to finding the real radical or other representations for the ideal should be investigated further. Also, for large number of variables, the currently used data structures do not scale well enough. Implementing the presented methods from Section 4.4 may improve the scalability.

Gröbner bases with respect to a lexicographic order are often used to support variable elimination, see e.g. [CLO97, Chapter 3]. Therefore, the computation and passing of such Gröbner basis might lead to much better performance of the VS and CAD module, as variable elimination gets a lot easier. In the current implementation, we are not able to investigate this, as the Gröbner module is very slow computing these. The longer running times are partly due to the structure of lexicographic Gröbner bases, and partly originated in the used strategies and data structures within the Buchberger algorithm.

In the last part of the thesis, we gave a description of the implementation of an extension to our module based on the Real Nullstellensatz. This approach seems to work well only if we are able to run in conflicts fast, as theory calls with a consistent set of polynomials always takes at least as long as with an inconsistent set. Moreover, if we do not transform inequalities into equalities, we miss a lot of conflicts during this procedure. However, those transformations worsen the performance of the other modules in SMT-RAT sharply, and the blow-up in the number of variables is currently also a problem for the application of the Nullstellensatz. However, a lot of simplifications are possible in order to reduce the number of variables. Implementing these may make the module applica-

ble. Despite the poor performance of the module, we learned some reasons to use numerical approaches within exact arithmetic, and as there are some other approaches which may benefit from numerical solutions, knowledge about the possibilities to use numerical libraries within exact arithmetic seems valuable. One noteworthy application is a fast check whether a polynomial can be rewritten as a sum of squares. This would simplify constraints with such polynomials a lot.

During development, a lot of ideas for improvements to the SMT framework were born. Some of them are already integrated. Tight integration of different modules seems to speed up the computation, but creates difficulties predicting how the theory solver behaves, which is a drawback when designing heuristics.

7.3 Future work

We see a lot of possibilities for future work, which can be classified into three main areas.

SMT and Gröbner bases

- Compute Gröbner bases which only have the same set of real zeroes, but not necessarily the same set of complex zeroes.
- Using Gröbner bases for elimination of variables, by applying a lexicographic order. Variable elimination [CLO97, Chapter 3] is one of the most eminent applications of Gröbner bases.
- Using Comprehensive Gröbner bases [Wei92],[Nab07] for case splitting or multivariate root counting [DSW98].
- By using activities, the order in which constraints are asserted can be influenced. Getting equalities early may help a lot.

Calculating Gröbner bases from SMT instances

- Using different (intelligent) variable orderings, as the ordering has a huge impact on the Gröbner basis and the computation time.
- Implementing additional strategies and data structures as discussed in Sections 4.2 and 4.4 and compare them on SMT benchmarks.
- Implementing the state-of-the-art algorithms as mentioned in Section 4.3, and compare them on SMT benchmarks.

Making the Real Nullstellensatz approach feasible

- Introduction of a better monomial iterator, maybe with activities on the variables (and a reduced number of variables).
- Reduce the number of constraint matrices which are constructed.
- Use an SDP library which supports incremental calls.
- Move to a dedicated module, which can be run in parallel.

Bibliography

[ÁCLS10] Erika Ábrahám, Florian Corzilius, Ulrich Loup, and Thomas Sturm. A lazy SMT-solver for a non-linear subset of real algebra. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (SMT '10)*, 2010.

[AL94] William. W. Adams and Philippe Loustaunau. *An Introduction to Gröbner Bases*. Graduate Studies in Mathematics. American Mathematical Society, 1994.

[AP10] Behzad Akbarpour and Lawrence Charles Paulson. Metitarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reason.*, 44(3):175–205, March 2010.

[Arm03] Armin Biere and Alessandro Cimatti and Edmund M. Clarke and Ofer Strichman and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

[BCR11] Anna Maria Bigatti, Massimo Caboara, and Lorenzo Robbiano. Computing inhomogeneous Gröbner bases. *J. Symb. Comput.*, 46(5):498–510, 2011.

[BHvMW09] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[Bor99] Brian Borchers. CSDP, A C library for semidefinite programming. *Optimization Methods and Software*, 11(1):613–623, 1999.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics. Springer New York, 2006.

[Bri10] Michael Brickenstein. Slimgb: Gröbner bases with slim polynomials. *Revista Matemática Complutense*, 23:453–466, 2010.

[BS98] Olaf Bachmann and Hans Schönemann. Monomial representations for Gröbner bases computations. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ISSAC '98, pages 309–316, New York, 1998. ACM.

[BT07] Clark Barrett and Cesare Tinelli. CVC3. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer Berlin, Heidelberg, July 2007.

[Buc65] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, 1965.

[Buc79] Bruno Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner-bases. In *Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Springer Berlin, Heidelberg, 1979.

[Buc85] Bruno Buchberger. Basic features and development of the critical-pair/completion procedure. In *Rewriting Techniques and Applications*, volume 202 of *Lecture Notes in Computer Science*, pages 1–45. Springer Berlin, 1985.

[BWK93] Thomas Becker, Volker Weispfenning, and Heinz Kredel. *Gröbner bases: a computational approach to commutative algebra*. Graduate texts in mathematics. Springer-Verlag, 1993.

[CA11] Florian Corzilius and Erika Ábrahám. Virtual substitution for SMT-solving. In *Proceedings of the 18th international conference on Fundamentals of Computation Theory*, FCT'11, pages 360–371. Springer Berlin, Heidelberg, 2011.

[CLJÁ] Florian Corzilius, Ulrich Loup, Sebastian Junges, and Erika Ábrahám. SMT-RAT manual. Available at http://smtrat.sourceforge.net/.

[CLJÁ12] Florian Corzilius, Ulrich Loup, Sebastian Junges, and Erika Ábrahám. SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox (tool presentation). In *Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'12)*, volume 7317 of *Lecture Notes in Computer Science*, pages 442–448. Springer, 2012.

[CLO97] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2nd ed.)*. Undergraduate texts in mathematics. Springer, 1997.

[CMT12] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. SMT-based scenario verification for hybrid systems. *Formal Methods in System Design*, pages 1–21, 2012.

[Col75] George Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20âĂŞ23, 1975*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer Berlin, Heidelberg, 1975.

[DGPS12] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann. SINGULAR 3-1-5 — A computer algebra system for polynomial computations. 2012. http://www.singular.uni-kl.de.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (ETAPS/TACAS'08)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[dMP09] Leonardo de Moura and Grant Olney Passmore. On locally minimal Nullstellensatz proofs. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, SMT '09, pages 35–42, 2009.

[DS97a] Andreas Dolzmann and Thomas Sturm. Redlog: computer algebra meets computer logic. *SIGSAM Bull.*, 31(2):2–9, 1997.

[DS97b] Andreas Dolzmann and Thomas Sturm. Simplification of quantifier-free formulas over ordered fields. *Journal of Symbolic Computation*, 24:209–231, 1997.

[DSW98] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real quantifier elimination in practice. In *Algorithmic Algebra and Number Theory*, pages 221–247. Springer, 1998.

[EP11] Christian Eder and John Edward Perry. Signature-based algorithms to compute Gröbner bases. In *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, ISSAC '11, pages 99–106, New York, 2011. ACM.

[ES04] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 333–336. Springer Berlin, Heidelberg, 2004.

[Fat03] Richard Fateman. Comparing the speed of programs for sparse polynomial multiplication. *SIGSAM Bull.*, 37(1):4–15, March 2003.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, 2002. ACM.

[FFK+08] Katsuki Fujisawa, Mituhiro Fukuda, Masakazu Kojima, Kazuhide Nakata, Maho Nakata, and Makoto Yamashita. SDPA (SemiDefinite Programming Algorithm) – User's manual. Technical report, Tokyo Institute of Technology, 2008.

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.

[GGV10] Shuhong Gao, Yinhua Guan, and Frank Volny, IV. A new incremental algorithm for computing Groebner bases. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 13–19, New York, 2010. ACM.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, 2nd edition, 1994.

[GM88] Rüdiger Gebauer and H. Michael Möller. On an installation of Buchberger's algorithm. *J. Symb. Comput.*, 6(2-3):275–286, December 1988.

[GMN+91] Alessandro Giovini, Teo Mora, Gianfranco Niesi, Lorenzo Robbiano, and Carlo Traverso. "One sugar cube, please" or selection strategies in the Buchberger algorithm. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, ISSAC '91, pages 49–54, New York, 1991. ACM.

[GS] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, 3rd edition, October 1996.

[Har07] John Harrison. Verifying nonlinear real formulas via sums of squares. In *Theorem Proving in Higher Order Logics, TPHOLs 2007*, volume 4732 of *Lecture Notes in Computer Science*, pages 102–118. Springer Berlin, Heidelberg, 2007.

[JdM12] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

[Joh74] Stephen C. Johnson. Sparse polynomial arithmetic. *SIGSAM Bull.*, 8(3):63–71, August 1974.

[KS08] Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer, 1 edition, 2008.

[MMT92] H. Michael Möller, Teo Mora, and Carlo Traverso. Gröbner bases computation using syzygies. In *Papers from the International Symposium on Symbolic and Algebraic Computation*, ISSAC '92, pages 320–328, New York, 1992. ACM.

[Mot67] Theodore S. Motzkin. The arithmetic-geometric inequality. In *Inequalities*, pages 205–224. New York: Academic Press, 1967.

[MP07] Michael Monagan and Roman Pearce. Polynomial division using dynamic arrays, heaps, and packed exponent vectors. In Victor Ganzha, Ernst Mayr, and Evgenii Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, volume 4770 of *Lecture Notes in Computer Science*, pages 295–315. Springer Berlin, Heidelberg, 2007.

[MP11] Michael Monagan and Roman Pearce. Sparse polynomial division using a heap. *Journal of Symbolic Computation*, 46(7):807 – 822, 2011.

[Nab07] Katsusuke Nabeshima. A speed-up of the algorithm for computing comprehensive gröbner systems. In *Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, ISSAC '07, pages 299–306, 2007.

[Par03] Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96:293–320, 2003.

[PdM09] Grant Olney Passmore and Leonardo de Moura. Superfluous S-polynomials in strategy-independent Groebner bases. In *11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) 2009*, pages 45–53, 2009.

[PdMJ10] Grant Olney Passmore, Leonardo de Moura, and Paul B. Jackson. Gröbner basis construction algorithms based on theorem proving saturation loops. In *Decision Procedures in Software, Hardware and Bioware*, number 10161 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

[PP08] Helfried Peyrl and Pablo A. Parrilo. Computing sum of squares decompositions with rational coefficients. *Theoretical Computer Science*, 409(2):269 – 281, 2008.

[PPdM12] Grant Passmore, Lawrence Paulson, and Leonardo de Moura. Real algebraic strategies for MetiTarski proofs. In *Intelligent Computer Mathematics*, volume 7362 of *Lecture Notes in Computer Science*, pages 358–370. Springer Berlin, Heidelberg, 2012.

[PQ08] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2008.

[PQR09] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In *Proceedings of the 22nd International Conference on Automated Deduction*, CADE-22, pages 485–501, Berlin, Heidelberg, 2009. Springer-Verlag.

[RS] Bjarke H. Roune and Michael Stillman. Practical Gröbner basis computation. In *Proceedings of the 2012 International Symposium on Symbolic and Algebraic Computation*, ISSAC '12. to appear.

[Ste74] Gilbert Stengle. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207:87–97, 1974.

[Tar51] Alfred Tarski. A decision method for elementary algebra and geometry. *Bulletin of the American Mathematical Society*, 59, 1951.

[Tiw05] Ashish Tiwari. An algebraic approach for the unsatisfiability of nonlinear constraints. In *Computer Science Logic*, volume 3634 of *Lecture Notes in Computer Science*, pages 248–262. Springer Berlin, Heidelberg, 2005.

[Tse83] Grigorii S. Tseitin. On the complexity of proofs in propositional logics. In *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer Berlin, Heidelberg, 1983.

[VB94] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1994.

[Wei92] Volker Weispfenning. Comprehensive Gröbner bases. *J. Symb. Comput.*, 14:1–29, July 1992.

[Wei93] Volker Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *Applicable Algebra in Engineering, Communication and Computing*, 8:85–101, 1993.

[Yan98] Thomas Yan. The geobucket data structure for polynomials. *J. Symb. Comput.*, 25(3):285–293, March 1998.