The present work was submitted to the LuFG Theory of Hybrid Systems

MASTER OF SCIENCE THESIS

USING EQUATIONAL CONSTRAINTS IN AN INCREMENTAL CAD PROJECTION

Rebecca Haehn

Examiners:
Prof. Dr. Erika Ábrahám
Priv. Doz. Viktor Levandovskyy

Additional Advisor:
Gereon Kremer

Aachen, 31.03.2018

Abstract

The cylindrical algebraic decomposition (CAD) method is a decision procedure for real algebra also used in satisfiability-modulo-theories (SMT) solving. Its approach is to reduce the problem to the case of univariate polynomials by repeatedly applying a projection operator to the set of input polynomials. Since the algorithm was first introduced several propositions for improved projection operators were made, in particular trying to use equational constraints.

This master thesis extends the implementation of the CAD method in the SMT solver SMT-RAT by using equational constraints. In the course of this thesis the approach for this that is introduced in McCallum's paper "On Projection in CAD-Based Quantifier Elimination with Equational Constraint" is implemented and extended to more than one projection level. This is supposed to reduce the number of polynomials that need to be computed for the projection. The implementation is additionally incremental, which is desirable for the usage in SMT solving, where CADs for similar sets of polynomials have to be computed.

Acknowledgements

Many thanks to Gereon Kremer for the continuous help with this thesis. Furthermore I would like to express my gratitude towards Erika Ábrahám for the opportunity to write my thesis about this interesting topic and to Viktor Levandovskyy for agreeing to be my second supervisor. Also thanks to my family for always supporting me.

Contents

Chapter 1

Introduction

In the last decades humanity developed more and more computerized systems. People became used to being surrounded by technology and to interact with technical systems in their everyday lives. Many tasks are handed over to machines due to an increasing willingness to rely on and trust in technology. For example the development of self-driving cars is an indication for this. However, with increasing complexity and potential risk of the tasks that are automated comes a high necessity for the correct functioning of such automated systems. The software that controls for example self-driving cars has to be highly reliable and bug-free. Therefore it is extensively tested during the development process. Testing is useful to prove that a software contains errors, but it is not suitable to prove the correctness of a program. For safety-critical applications the latter is desirable to ensure accurate functioning of the software.

The attempt to verify the correctness of a program on the basis of a mathematical calculus is called software verification. During this procedure the program is typically transformed into a mathematical model and the properties to be proven are formalized on the basis of this model. An example for a powerful modeling formalism is first-order-logic. When using this formalism, the program can be verified by deciding whether a corresponding first-order-logic formula in the context of a background theory is satisfiable. Depending on the application different background theories can be used. Problems of this type are called satisfiability-modulo-theories (SMT) problems. These problems are quite complex to solve, the decision problem for propositional logic formulas is already NP-complete. Still some programs, so called SMT solvers, aim to solve them, since despite bad worst-case complexity many problems can be solved reasonably fast. Nevertheless, it is always strived to further improve the used methods.

One example for an SMT solver is SMT-RAT, which is maintained as an open source project by the Theory of Hybrid Systems research group at RWTH Aachen University. SMT-RAT combines a satisfiability problem (SAT) solver and theory solvers for different background theories in order to solve SMT problems. For example for the theory of real numbers SMT-RAT contains an implementation of the cylindrical algebraic decomposition (CAD) method, which decomposes $\mathbb{R}^n$ according to a set of polynomials into cylindrically arranged cells. The approach of this method is to reduce the problem to the case of univariate polynomials by repeatedly applying a projection operator to the initial set of polynomials. To apply such a projection operators is computationally expensive, therefore, in order to speed up the computation, several improvements for the projection operator were proposed. In this thesis the so called restricted projection operator McCallum presented in [16] is considered. This operator makes use of equational constraints to reduce the number of polynomials that have to be computed.

In the course of this thesis the restricted projection operator is added to the implementation of the CAD method in SMT-RAT. It has to be taken into account that the projection is computed incrementally in this implementation, which might reduce the impact of this operator on the computational effort. In the attempt to further improve the CAD method the restricted operator is not only used in the first level, but in as many consecutive levels (starting with the first) as possible, despite the usage of it only being validated for the first level unless the CAD is computed for $\mathbb{R}^3$. It is going to be examined whether the application of this operator in successive levels causes errors on a set of benchmarks. Further examinations consider the impact of this change on the overall performance of SMT-RAT.

Two other contributions to the CAD method in SMT-RAT are implemented as well. One is the use of bounds for individual variables to simplify the projection, the other is to deactivate polynomials instead of deleting them. Afterwards different combinations of these three changes in the implementation are evaluated, to determine whether they improve the overall performance of the CAD implementation.

This thesis begins with an introduction to three decision procedures for different problems in Chapter 2. First propositional logic formulas and SAT are described, followed by the DPLL algorithm, which is used to this kind of problems. It is continued with an introduction to the CAD method, which is used to solve non-linear real algebra problems. This method consists of two phases, the projection phase, which is described in more detail since it is required for the main part of this thesis, and the lifting phase. Next it is explained how these two decision procedures can be combined in SMT solvers in order to solve combinations of the previously introduced problems. Then the SMT solver SMT-RAT is introduced with focus on the incrementality of its components. In Chapter 3 the restricted projection operator mentioned before is explained as well as a slightly different semi-restricted projection operator. The implementation of the former is described in Chapter 4 after a description of the other two changes in the implementation, namely the simplification by using bounds and the deactivation of polynomials. The experimental results for evaluating the performance of the changed implementation on some benchmarks are presented and discussed in Chapter 5. And finally a summary of this thesis as well as a short outline of possible future work is given in Chapter 6.

Chapter 2

Preliminaries

In this chapter first the *satisfiability problem (SAT)* is introduced as well as the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm, which is implemented in most SAT solvers. Then the *cylindrical algebraic decomposition (CAD)* method, a decision procedure for real algebra, is described. This method consists of a projection and a lifting phase. For a better understanding of the main part of this thesis the projection phase is explained in more detail. It is continued with an introduction to *satisfiability-modulo-theories (SMT)* solvers that allow to combine these two methods followed by a description of one of these, namely, SMT-RAT.

2.1 Satisfiability checking

The *satisfiability problem (SAT)* is to decide whether a given propositional logic formula is satisfiable. Such a formula is composed of propositions, negations $\neg$, disjunctions $\vee$, conjunctions $\wedge$ and implications $\Rightarrow$. For all propositional logic formulas exists an equisatisfiable formula in *conjunctive normal form (CNF)* (see Definition 2.1.1) that can be generated with only linear growth in size by using Tseitin's transformation [20]. Therefore in the following it is assumed that the occurring propositional logic formulas are in CNF.

Definition 2.1.1 (CNF).
A formula φ of the form

$$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} p_{ij},$$

with literals p_{ij}, where a literal is either a proposition or its negation, is in CNF. The disjunctions of literals are called clauses.

A propositional logic formula φ is satisfiable if a variable assignment $\alpha : \{p_1,...,p_n\} \rightarrow \{true,\ false\}$ for the propositions p_1, ..., p_n in φ exists such that the formula evaluates to $true$ under the usual interpretation of the connectives.

SAT is an NP-complete problem, which means that probably no algorithm exists that can solve it in polynomial-time. Nevertheless in practice there are some algorithms that perform well on many SAT instances despite bad worst-case complexity.

Algorithm 1 The DPLL algorithm

```
function DPLL(propositional logic formula $\varphi$)
    while true do
        while $\neg$BCP($\varphi$) do
            if $\neg$RESOLVE() then
                return unsat
            end if
        end while
        if $\neg$DECIDE($\varphi$) then
            return sat
        end if
    end while
end function
```

One of these is the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm presented in [11] and [10] that is implemented in many state-of-the-art SAT solvers.

The DPLL algorithm (in Algorithm 1) is illustrated in Figure 2.1 for a better understanding. Its approach is to iteratively build a partial truth assignment starting with only unevaluated propositional variables. Therefore first *Boolean constraint propagation (BCP)* is executed. This method examines whether under the current partial assignment of truth values to propositions other assignments are implied in order for the formula to evaluate to *true*. In case there occurs a conflict, which means that a variable needs to be assigned *true* and $false$ for the formula to be satisfied BCP returns $false$. In that case RESOLVE_CONFLICT is executed, in this method conflict-driven clause-learning and backtracking are applied in order to resolve the conflict. If this is impossible the formula is unsatisfiable and $false$ is returned, otherwise the algorithm continues with BCP. When BCP can be executed without the occurrence of a conflict it is checked whether all variables have a truth value assigned. If that is the case the formula is satisfiable and DECIDE returns $false$, otherwise a decision is made by DECIDE. Making a decision means to choose a variable without truth value and assign one. After a decision BCP is applied and the loop described above is executed again until the satisfiability of the formula is determined. An example for the execution of the DPLL algorithm can be found in Example 2.1.1.

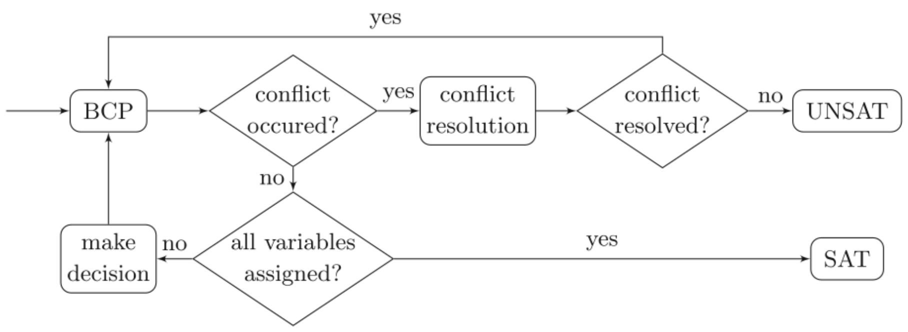


Figure 2.1: DPLL [1]

Example 2.1.1 (DPLL algorithm).

```
Input: $\varphi = c_1 \wedge c_2 \wedge c_3 \wedge c_4 = p_1 \wedge (p_2 \vee p_3) \wedge (\neg p_3 \vee p_4) \wedge (\neg p_1 \vee p_2 \vee \neg p_4)$;
  $\alpha = \emptyset$
  BCP:     $\alpha \leftarrow \alpha \cup \{p_1 \mapsto true\}$                          ▷ implied by $c_1$
  DECIDE:  $\alpha \leftarrow \alpha \cup \{p_2 \mapsto false\}$
  BCP:     $\alpha \leftarrow \alpha \cup \{p_3 \mapsto true\}$                          ▷ implied by $c_2$
  BCP:     $\alpha \leftarrow \alpha \cup \{p_4 \mapsto true\}$                          ▷ implied by $c_3$
  Conflict: $c_4$
  CONFLICT_RESOLUTION:
        $c_5 = Resolution(c_3,c_4) = \neg p_1 \vee p_2 \vee \neg p_3$                 ▷ conflict clause
        $\varphi \leftarrow \varphi \wedge c_5$
        $\alpha \leftarrow \alpha \setminus \{p_2 \mapsto false;\ p_3 \mapsto true;\ p_4 \mapsto true\}$    ▷ backtracking
  ...
  $\alpha = \{p_1 \mapsto true,\ p_2 \mapsto true,\ p_3 \mapsto true,\ p_4 \mapsto true\}$
  No conflict, all variables assigned.
Output: SAT
```

The performance of the DPLL algorithm depends on various details in the implementation for example the conflict resolution and the heuristic for choosing the next variable when a decision is made. Since the general concept of the DPLL algorithm is sufficient for this thesis further improvements of the algorithm are not discussed here.

After introducing the decision problem for propositional logic formulas in this section, the next section is about the theory of real numbers and a decision procedure for real algebra.

2.2 Cylindrical algebraic decomposition

The *cylindrical algebraic decomposition (CAD)* method is a decision procedure for real algebra, more precisely a method for quantifier elimination to determine whether a first-order-logic formula has a solution regarding the underlying theory of real numbers. In the context of this thesis the CAD method is only used for sets of constraints. This satisfiability problem is decidable [19], but the CAD method has doubly exponential complexity in the number of variables [6]. In the following the theory of real numbers "Reals" is defined as in the SMT-LIB [3] (see Definition 2.2.1) with the common semantics for the function symbols and comparison predicates. The function symbol $-$ represents actually two functions, the unary negation and the binary subtraction.

Definition 2.2.1 (Theory of real numbers "Reals").

domain: $\mathbb{R}$

function symbols: $-\ ,\ +\ ,\ \cdot\ ,\ /$

comparison predicates: $\leqslant\ ,\ <\ ,\ \geqslant\ ,\ >$

A polynomial over the real numbers (see Definition 2.2.3) consists of coefficients in $\mathbb{R}$, variables and the function symbols $+$ and $\cdot$. A polynomial is the sum of monomials (see Definition 2.2.2). The corresponding polynomial function for a polynomial p is called p as well.

Definition 2.2.2 (Monomial).
A term of the form

$$M = a \cdot \prod_{i=1}^{n} x_i^{e_i},$$

with $e_i \in \mathbb{N}_0$ and a coefficient $a \in \mathbb{R}$, is called monomial.

Definition 2.2.3 (Polynomial).
A term of the form

$$p = \sum_{k=1}^{m} M_k,$$

with monomials M_k, is called polynomial.

The set of all polynomials with coefficients in $\mathbb{R}$ and variables x_1, ..., x_n is denoted as $\mathbb{R}[x_1,...,x_n]$. Polynomials in $\mathbb{R}[x_i]$ for some variable x_i are called univariate. Multivariate polynomials are those in $\mathbb{R}[x_1,...,x_n]$ for $n > 1$, they can be interpreted as univariate polynomials in $\mathbb{R}[x_1,...,x_{n-1}][x_n]$. These multivariate polynomials are then used to define so called quantifier-free non-linear real arithmetic constraints. Such a constraint (see Definition 2.2.4) is a polynomial compared to zero.

Definition 2.2.4 (Polynomial constraint).
An expression of the form

$$p \Diamond 0,$$

with $p \in \mathbb{R}[x_1,...,x_n]$ and $\Diamond \in \{<, \leqslant, =, \neq, \geqslant, >\}$, is called polynomial constraint.

In this thesis the purpose of the CAD method is to determine whether a set (or conjunction) of constraints $\mathcal{C}$ is satisfiable. This is the case if a point $a = (a_1,...,a_n)$ in $\mathbb{R}^n$ exists such that all constraints in the set are satisfied when the corresponding polynomial functions are evaluated in a. Another application of the CAD method is quantifier elimination, which is not covered in this thesis.

The CAD method is a fundamental algorithm in the field of real algebra and was first introduced by Arnon, Collins and McCallum in [2]. The approach of this algorithm is to decompose $\mathbb{R}^n$ into finitely many cylindrically arranged cells, each represented by one sample point, such that the solution for the decision problem can be determined by evaluating the polynomial functions only in these sample points.

Definition 2.2.5 (Decomposition).
A set

$$\{C_1, ..., C_m\}$$

is a decomposition of $\mathbb{R}^n$, if $\mathbb{R}^n = \biguplus_{i=1}^{m} C_i$ holds.

Decompositions are introduced in Definition 2.2.5. A decomposition is algebraic if all of its cells are semi-algebraic sets, which means they can be described by a conjunction of polynomial constraints. The cells are cylindrically arranged when the projections of two cells onto a lower dimension are either equal or disjoint. These definitions are taken from [2]. A sign-invariant CAD (see Definition 2.2.6) is suitable for the above described purpose.

Definition 2.2.6 (Sign-invariance).
Given $A \subset \mathbb{R}^n$ and $p \in \mathbb{R}[x_1,...,x_n]$, p is called sign-invariant on A, if either

- $p(x) > 0 \ \forall \ x \in A$ *or*
- $p(x) = 0 \ \forall \ x \in A$ *or*
- $p(x) < 0 \ \forall \ x \in A$ *holds.*

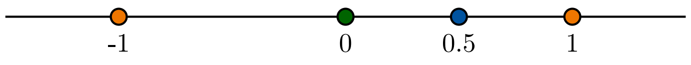
$P \subset \mathbb{R}[x_1,...,x_n]$ is called sign-invariant on A, if each $p \in P$ is sign-invariant on A. P is furthermore sign-invariant on $C = \{c_1,...,c_m\}$, with $c_i \subset \mathbb{R}^n$, if P is sign-invariant on c_i for all $1 \leqslant i \leqslant m$.

Definition 2.2.7 (Real root).
$\alpha \in \mathbb{R}^n$ is a real root of the polynomial $p \in \mathbb{R}[x_1,...,x_n]$ iff $p(\alpha) = 0$.

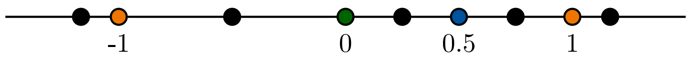
A sign-invariant CAD is determined by the roots of the polynomials in the input set. Since the real roots (see Definition 2.2.7) of univariate polynomials can be computed effectively, a sign-invariant CAD for $\mathbb{R}$ can be computed as is illustrated in the Example 2.2.1. The idea behind the CAD method is, that multivariate polynomials are projected to univariate polynomials by eliminating the variables one after another. Then a CAD on $\mathbb{R}$ is determined and lifted to a CAD of $\mathbb{R}^n$.

Example 2.2.1 (One dimensional CAD).
Let a set of polynomials be given by $\{p_1 = x_1 - 0.5;\ p_2 = -2 \cdot x_1^2;\ p_3 = x_1^2 - 1\}$

- *Real roots of the polynomials:*



- *CAD:* $(-\infty, -1)$, -1, $(-1,0)$, 0, $(0,0.5)$, 0.5, $(0.5,1)$, 1, $(1,\infty)$
- *Sample points:*



So to construct a sign-invariant CAD two phases are executed: the projection and the lifting phase. During the projection phase a projection operator that decreases the number of variables by one is applied repeatedly until the remaining polynomials are univariate. During this phase the polynomials that describe the boundaries of the CAD cells are computed. Afterwards are in the lifting phase the sample points generated by computing the real roots of only univariate polynomials. These two phases are described in more detail in the next two subsections. Especially the projection phase is important for the main part of this thesis.

2.2.1 Projection

A projection operator P has to have the property that for a set of polynomials $\mathcal{P}_n$ each CAD $\mathcal{C}_{n-1}$ for $P(\mathcal{P}_n)$ can be extended to a CAD $\mathcal{C}_n$ for $\mathcal{P}_n$ by constructing the cells of $\mathcal{C}_n$ to be the sign-invariant regions for all polynomials in $\mathcal{P}_n$ in the cylinders $C_{n-1,k} \times \mathbb{R}$ for each $C_{n-1,k}$ in $\mathcal{C}_{n-1}$. This property is required for lifting.

Definition 2.2.8 (Delineability).
A polynomial $p \in \mathbb{R}[x_1,...,x_n]$ is delineable over a connected region $C \subset \mathbb{R}^{n-1}$, if for all $x \in C$ holds $p(x,x_n)$ is identically zero and the real roots of p over C define continuous real-valued functions θ_1, ..., θ_s such that for all $x \in C$ $\theta_1(x) < ... < \theta_s(x)$ and for each θ_i there is an $m_i \in \mathbb{N}$ such that m_i is the multiplicity of the root $\theta_i(x)$ of $p(x,x_n)$.

To be able to decompose a region $C \times \mathbb{R}$ to sign-invariant cylindrically arranged cells for a set of polynomials $\mathcal{P}$ the polynomials in $\mathcal{P}$ have to be delineable over C (see Definition 2.2.8 according to [7]). Delineability is needed to define the boundaries of cylindrical cells.

There are several different projection operators that have the above mentioned property, the best known are Collins' [2] and Hong's [13]. For McCallum's [14] and Brown's [7] projection operator this can be for certain inputs achieved. These projection operators were compared in [21] where McCallum's operator proved to be far better than Collins', despite being incomplete. Hong's operator was suggested as a possibility for cases where McCallum's operator is not applicable, since the number of polynomials is significantly reduced compared to Collins', however generally not as much as when using McCallum's operator. Brown's and McCallum's operators achieved similar results. However they still have the disadvantage that the number of projected polynomials grows exponentially.

In this thesis Brown's projection operator as presented in [7] is used, since the examination in [21] found that Brown's and McCallum's operator are working comparably well and when using Brown's projection operator the resulting projection contains less polynomials. Brown's operator also keeps the amount of lifting points smaller. Otherwise the two operators are quite similar. Before formally defining Brown's projection operator some mathematical definitions that are required for computing the projection are introduced.

In the following a multivariate polynomial p in $\mathbb{R}[x_1,...,x_n]$ is interpreted as univariate polynomial in $\mathbb{R}[x_1,...,x_{n-1}][x_n]$. With this interpretation it is possible to define the leading coefficient for multivariate polynomials, see Definition 2.2.9.

Definition 2.2.9 (Leading coefficient).
The leading coefficient of a polynomial $p = \sum_{i=0}^{m} p_i \cdot x_n^i \in \mathbb{R}[x_1,...,x_{n-1}][x_n]$ is defined as

$$lcf(p) = p_m \in \mathbb{R}[x_1,...,x_{n-1}].$$

Apart from the leading coefficient also discriminants and resultants are required for the projection operator. These are defined using the Sylvester matrix (see Definition 2.2.10). The following mathematical foundations are taken from [5].

Definition 2.2.10 (Sylvester matrix).
The matrix

$$Syl(p,q) \;=\; \begin{pmatrix} a_k & & \dots & & a_0 & & & \\ & a_k & & \dots & & a_0 & & \\ & & \ddots & & & & \ddots & \\ & & & a_k & & \dots & & a_0 \\ b_l & & \dots & & b_0 & & & \\ & b_l & & \dots & & b_0 & & \\ & & \ddots & & & & \ddots & \\ & & & b_l & & \dots & & b_0 \end{pmatrix},$$

for polynomials $p = \sum_{i=0}^{k} a_i \cdot x_n^i$ *and* $q = \sum_{j=0}^{l} b_j \cdot x_n^j$*, is called Sylvester matrix.*

Resultants and discriminants are then defined in Definition 2.2.11. Both can be defined using the roots of p and q. This definition shows that resultants can be used to check whether two polynomials p and q have common roots since they vanish exactly in that case, while the discriminant of a polynomial p vanishes if and only if p has a multiple root. Both can be computed as the determinant of a Sylvester matrix.

Definition 2.2.11 (Resultants and discriminants).
For polynomials $p = \sum_{i=0}^{m} a_i \cdot x_n^i$ *and* $q = \sum_{j=0}^{l} b_j \cdot x_n^j$*, with the roots* x_1, ..., x_m *respectively* y_1, ..., y_l*, is the resultant defined as*

$$res(p,q) \;=\; a_m^l \cdot b_l^m \cdot \prod_{1 \leqslant i \leqslant m} \prod_{1 \leqslant j \leqslant l} (x_i - y_j) \;=\; det(Syl(p,q))$$

and the discriminant as

$$disc(p) \;=\; \prod_{1 \leqslant i < j \leqslant m} (x_i - x_j)^2 \;=\; det(Syl(p,p')),$$

where p' is the derivative of p.

Given the necessary definitions to compute the projection using Brown's operator, next the requirements for using this operator are described. As well as McCallum's operator, described in [14], Brown's operator requires the use of the finest square-free basis (see Definition 2.2.13), in each projection step.

Definition 2.2.12 (Square-free).
A polynomial $p \in \mathbb{Z}[x_1,..,x_n]$ *is called square-free, if* q^2 *does not divide* p*,* $\forall q \in \mathbb{Z}[x_1,..,x_n] \backslash \mathbb{Z}[x_1,..,x_{n-1}]$.

Definition 2.2.13 (Finest square-free basis).
A finest square-free basis of a set of polynomials $P = \{p_1, ..., p_m\} \subset \mathbb{Z}[x_1,..,x_n]$ *is the set of all irreducible factors of* $\prod_{i=1}^{m} p_i$.
A set of pairwise relatively prime irreducible integral polynomials is also referred to as finest square-free basis.

The correctness of the CAD can also only be guaranteed when the input polynomials are well-oriented (see Definition 2.2.16, [15]), otherwise projection factors might vanish identically over a region of dimension greater than zero. McCallum argued that the vast majority of polynomials is well-oriented, particularly all polynomials with at most three variables. Also there is some modification to the lifting stage that allows the CAD to stay correct in some cases, even if the polynomials are not well-oriented.

Definition 2.2.14 (Content and primitive part of a polynomial).
For $f = \sum_{i=0}^{m} a_i \cdot x_n^i \in \mathbb{R}[x_1,...,x_n]$ *with* $a_i \in \mathbb{R}[x_1,...,x_{n-1}]$ *and* $a_m \neq 0$ *is*

- $cont(f) = gcd(a_0,...,a_m)$ *the content, when gcd is the greatest common divisor.*
- $prim(f) = f/cont(f)$ *the primitive part.*

Definition 2.2.15 (Content and primitive part of a set of polynomials).
For $A \subset \mathbb{R}[x_1,...,x_n]$ *is*

- $cont(A) = \{cont(f) | f \in A$ *and* $cont(f)$ *is non-zero, non-unit*$\}$ *the content of* A,
- $prim(A) = \{prim(f) | f \in A$ *and* $prim(f)$ *has positive degree*$\}$ *the primitive part of* A,

with $cont(f)$ *and* $prim(f)$ *for* $f \in \mathbb{R}[x_1,...,x_n]$ *as in Definition 2.2.14.*

Definition 2.2.16 (Well-oriented).
Let $prim(A)$ *and* $cont(A)$ *be given as in Definition 2.2.15.*
A set $A \subset \mathbb{R}[x_1,...,x_n]$ *is called well-oriented with regard to a projection operator* P, *if no element of* $prim(A)$ *vanishes on any sub-manifold of* $\mathbb{R}^{n-1}$ *of positive dimension and* $cont(A) \cup \mathcal{P}(B)$, *where* B *is the finest square-free basis for* $prim(A)$, *is well-oriented.*

For a set $\mathcal{P}_n$ of well-oriented polynomials with the finest square-free basis $\mathcal{P}'_n$ Brown's projection operator is defined in Definition 2.2.17, according to [6]. $\mathcal{P}_n$ is called projection level n. The projection operator is applied repeatedly until the entire projection $P = \mathcal{P}_n \cup ... \cup \mathcal{P}_1$ is computed. In this process the variables are regarded in descending order. It is first projected with respect to x_n and in the last level the polynomials are univariate in x_1. The greatest variable present in a polynomial p, with respect to the ordering, is called main variable of p.

Definition 2.2.17 (Brown's projection operator).

$$\begin{aligned} Proj(\mathcal{P}_n) &= \mathcal{P}_{n-1} \\ &= \{lcf(p_i),\ disc(p_i),\ res(p_i,p_j) \mid p_i,\ p_j \in \mathcal{P}'_n, i \neq j\} \ \cup cont(\mathcal{P}_n) \\ &\subset \mathbb{R}[x_1,...,x_{n-1}], \end{aligned}$$

for $\mathcal{P}_n \subset \mathbb{R}[x_1,...,x_n]$ *and* $\mathcal{P}'_n$ *finest square-free basis of* $\mathcal{P}_n$.

Due to the resultants of pairwise distinct polynomials the projected polynomials grow quadratically in number and size. It should be noted that the polynomials in $\mathcal{P}_1$ are univariate, which means that their roots can be computed efficiently. This is used in the lifting phase of the CAD method to generate the CAD.

2.2.2 Lifting

In the lifting phase the projection levels, generated in the projection phase and described in the previous section, are used in reverse order to compute the CAD. This procedure is outlined below, as explained in [7]:

- compute $\mathcal{C}_1$ (CAD of $\mathbb{R}^1$) defined by $\mathcal{P}_1$

- compute $\mathcal{C}_k$ (CAD of $\mathbb{R}^k$) defined by $\mathcal{P}_1 \cup ... \cup \mathcal{P}_k$ by lifting over cells of $\mathcal{C}_{k-1}$
- $C \in \mathcal{C}_{k-1}$ is represented by $(\alpha_1,...,\alpha_{k-1}) \in C$, substitute $(\alpha_1,...,\alpha_{k-1})$ for $x_1,...,x_{k-1}$ in $\mathcal{P}_k$ and compute the roots of the resulting polynomials univariate in x_k to extend $\mathcal{C}_{k-1}$

It is started with the first projection level $\mathcal{P}_1$, since this level contains only univariate polynomials. First the real roots of the polynomials in $\mathcal{P}_1$ are computed. Then the CAD $\mathcal{C}_1$ is the set that contains the real roots and the intervals in-between as well as the intervals from $-\infty$ to the smallest root and from the greatest root to ∞. $\mathcal{C}_1$ is represented by a set of sample points. The real roots are represented by themselves and the intervals each by one point in the interval. This was already illustrated in Example 2.2.1. Then the property of the projection operator described in Section 2.2 is used and the CAD $\mathcal{C}_1$ of $\mathbb{R}$ is extended to a CAD $\mathcal{C}_2$ of $\mathbb{R}^2$ and so on.

Let the CAD $\mathcal{C}_{n-1} = \{C_1,...,C_m\}$ of $\mathbb{R}^{n-1}$ be represented by the sample points $\{\alpha_1,...,\alpha_m\} \subset \mathbb{R}^{n-1}$. Then it is extended to $\mathcal{C}_n$ in the following way: $(x_1,...,x_{n-1})$ is substituted by α_i in $\mathcal{P}_n$, the resulting set of polynomials is called $\mathcal{P}_n[\alpha_i]$. Next the roots of the univariate polynomials in $\mathcal{P}_n[\alpha_i]$ are computed. These are used to decompose $C_i \times \mathbb{R}$ into sign-invariant regions for the polynomials in $\mathcal{P}_n$. Each of these regions is represented by (α_i, β), for $\beta \in \mathbb{R}$ representing one of the regions in the one-dimensional CAD of $\mathcal{P}_n[\alpha_i]$. This is done for all $C_i \in \mathcal{C}_{n-1}$, $\mathcal{C}_n$ is then composed of all the resulting cells.

Since Brown's projection operator is used it can happen that some polynomials in the projection are not delineable, which is required to be able to decompose a region into cylindrical cells. An example for non-delineable polynomials over a region C is given in Example 2.2.2. The set of polynomials used there is however delineable over each of the regions C_1, ..., C_5. This shows that delineability may be achieved by further decomposing the cell.

Example 2.2.2 (Delineability).

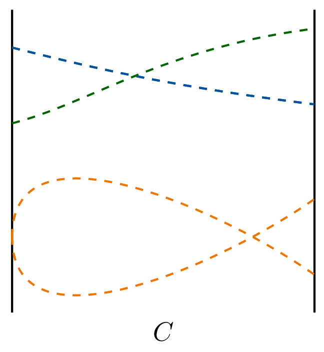


Figure 2.2: Not delineable

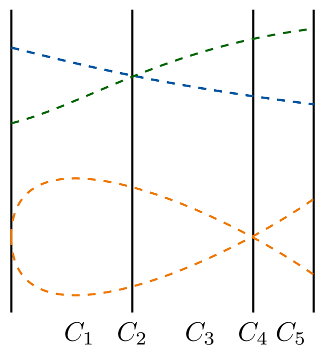


Figure 2.3: Delineable

This is relevant when using Brown's projection operator, because it might happen that projection factors vanish identically over finitely many isolated points. These points need to be computed and added to the CAD during the lifting phase (this has no impact on the projection phase). Often some of these points are already zero-dimensional cells in the CAD without being explicitly added. So when a polynomial $p \in \mathcal{P}_n$ vanishes identically over a cell in $\mathcal{C}_{n-1}$, p is replaced with a so called delineating polynomial for lifting, which ensures the delineability of the other polynomials. More details about this process can be found in [7] and [22].

2.3 Satisfiability-modulo-theories solver

In Section 2.1 a decision procedure for propositional logic formulas is introduced and in Section 2.2 one for real algebra. These two procedures can be combined in a *satisfiability-modulo-theories (SMT)* solver. What kind of problems can be solved that way and how is described in this section.

An SMT solver is a computer program with the purpose to decide the satisfiability of an SMT formula, more specifically to decide whether a given first-order logic formula is satisfiable with respect to combinations of background theories. Such a decision problem is called SMT problem. In the SMT-LIB standard [3] precise definitions of the syntax and semantics of specific classes of SMT problems are established. Therefore it is distinguished between logics and theories that can be combined as desired. In the course of this thesis the logic of non-linear real arithmetic, which is based on the theory of real numbers, is of interest, though restricted by the exclusion of quantifiers. This subclass of SMT problems is called *quantifier-free non-linear real arithmetic (QF_NRA)* in the SMT-LIB. The specific problem is then to decide whether there exist real values for the variables in a non-linear real arithmetic formula such that the formula is satisfied. Due to the restriction that the logical formula contains no quantifiers this is a generalization of SAT (see Section 2.1) by using polynomial constraints (see Definition 2.2.4) instead of propositional variables. So QF_NRA problems are the combination of the two decision problems presented in the previous sections.

Next SMT solvers are described, for more details see [4]. In this thesis only SMT solvers constructed according to the so called lazy SMT solving approach, outlined in Figure 2.4, are considered. A lazy SMT solver combines a SAT solver and theory solvers (for conjunctions of constraints).

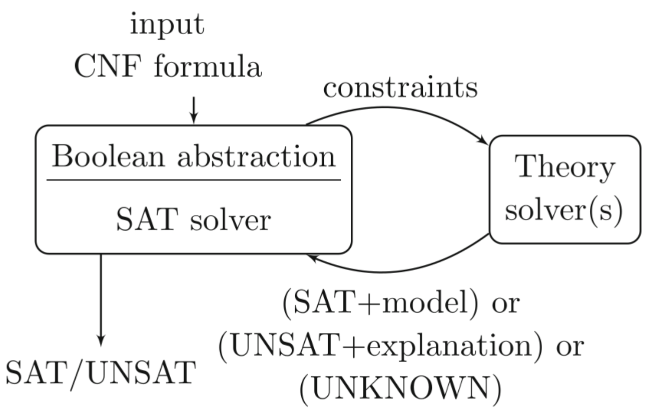


Figure 2.4: SMT solving framework [1]

Algorithm 2 The $\mathcal{T}$-DPLL algorithm

```
function $\mathcal{T}$-DPLL(quantifier-free SMT formula $\varphi$)
    $\varphi \leftarrow skel(\varphi)$
    if $\neg$BCP($\varphi$) then
        return $unsat$
    end if
    while $true$ do
        if $\neg$DECIDE($\varphi$) then
            return $sat$
        end if
        repeat
            while $\neg$BCP($\varphi$) do
                if $\neg$RESOLVE_CONFLICT() then
                    return $unsat$
                end if
            end while
            $t \leftarrow \mathcal{T}$-DEDUCTION()
            $\varphi \leftarrow \varphi \wedge skel(t)$
        until $t \equiv true$
    end while
end function
```

First the included SAT solver computes the Boolean abstraction of the input SMT formula (see Definition 2.3.1), there an SMT formula is understood as a propositional logic formula with polynomial constraints instead of propositional variables. Afterwards it tries to find a solution for the Boolean abstraction. Then the theory solver gets a conjunction of constraints, corresponding to the current (partial) assignment for the propositional variables, as input and tries to find a solution for these [4]. In case no solution exists the theory solver returns an infeasible subset to the SAT solver that uses this to find another Boolean solution if possible, otherwise the theory solver returns the solution.

Definition 2.3.1 (Boolean abstraction)**.**
The Boolean abstraction $skel(\varphi)$ of an SMT formula φ is the propositional logic formula that is obtained by replacing each constraint p in φ by a fresh propositional variable X_p.

It is further distinguished between full and less lazy approaches. In the full lazy approach the SAT solver tries to find a full solution for the Boolean abstraction before the theory solver gets a conjunction of constraints and searches for a theory solution, while in the less lazy approach after every conflict-free BCP execution the theory consistency is checked on the partial assignment [1]. The less lazy approach has the advantage that theory inconsistent assignments can be detected earlier. And in case the result is *SAT* and the partial assignments are consistent with the theory the computational effort is not higher than that of the full lazy approach, at least when using an incremental theory solver (see Section 2.3.1). However when the Boolean abstraction is already unsatisfiable with the less lazy approach some unnecessary theory calls are made.

Less lazy SMT solvers implement the $\mathcal{T}$-DPLL algorithm as shown in Algorithm 2, which is an extension of the DPLL algorithm introduced in Section 2.1. For further details see [18]. The functions BCP, DECIDE and RESOLVE_CONFLICT are the same as used in the DPLL algorithm. What is different is that after every conflict-free BCP call the method $\mathcal{T}$-DEDUCTION is executed. This function first transforms the partial assignment according to Definition 2.3.2 to a set of theory constraints Φ.

Definition 2.3.2 (Theory solver input).
The theory solver input for a partial assignment α is

$$\Phi \ = \ \{p \mid \alpha(X_p) = true\} \cup \{\neg p \mid \alpha(X_p) = false\}.$$

Then it checks the consistency of the constraints in Φ by using a theory solver. $\mathcal{T}$-DEDUCTION returns *true* in case of consistency, otherwise an infeasible subset of Φ is computed and the negated conjunction of the constraints in this subset is returned. In the second case the returned clause is added to the initial SMT formula. Then the loop is repeatedly executed until either a solution or an explanation for the unsatisfiability of the input is found.

Using the propositional logic formula in Example 2.1.1 as the Boolean abstraction of an SMT formula an example for a $\mathcal{T}$-DPLL execution is given in Example 2.3.1.

Example 2.3.1 ($\mathcal{T}$-DPLL algorithm).
Input: $\varphi \ = \ p_1 \ \wedge \ (p_2 \vee p_3) \ \wedge \ (\neg p_3 \vee p_4) \ \wedge \ (\neg p_1 \vee p_2 \vee \neg p_4)$; $\alpha \ = \ \varnothing$
with p_1 *instead of* $x \neq 0$, p_2 *for* $y \neq 0$, p_3 *for* $x^2 \cdot y^2 = 0$ *and* p_4 *for* $x^2 - 1 < 0$

```
DPLL: $\alpha = \{p_1 \mapsto true\}$
$\mathcal{T}$-DEDUCTION: true for $x = 1$
DPLL: $\alpha = \{p_1 \mapsto true;\ p_2 \mapsto true;\ p_3 \mapsto true;\ p_4 \mapsto true\}$
$\mathcal{T}$-DEDUCTION:
        $\{p_1, p_2, p_3\}$                                   ▻ infeasible subset
        $c_6 = \neg p_1 \vee \neg p_2 \vee \neg p_3$
        $\varphi \leftarrow \varphi \wedge c_6$
        $\alpha \leftarrow \alpha \setminus \{p_3 \mapsto true;\ p_4 \mapsto true\}$        ▻ backtracking
...
DPLL: $\alpha = \{p_1 \mapsto true;\ p_2 \mapsto true;\ p_3 \mapsto false;\ p_4 \mapsto true\}$
$\mathcal{T}$-DEDUCTION: true for $x = 0.5;\ y = 1$
No conflict, all variables assigned.
```
Output: *SAT, for* $x \ = \ 0.5;\ y \ = \ 1$

An SMT solver implementing this approach is for example SMT-RAT, which is described in the next section.

2.3.1 SMT-RAT

SMT-RAT is the abbreviation for *satisfiability-modulo-theories real arithmetic toolbox*, which is an SMT solver written in C++. This solver is maintained as an open source project by the Theory of Hybrid Systems research group at RWTH Aachen University. It is described in [1], in more detail in [9], and includes a CAD implementation, which is used as basis for the implementation in this thesis.

The solver contains methods to solve quantifier-free (non-)linear real and integer arithmetic and quantifier-free formulas over the theory of fixed-size bitvectors [9]. For this thesis is important that SMT-RAT is suitable for QF_NRA as described above.

Algorithm 3 Adding polynomials in an incremental projection

```
1: function ADD_POLYNOMIAL(polynomial $p$)
2:     $\mathcal{P}_n \leftarrow \mathcal{P}_n \cup \{p\}$
3:     COMPLETE_PROJECTION($\{p\}$,$n$)
4: end function

5: function COMPLETE_PROJECTION(set of polynomials $\mathcal{P}$, index of main variable $n$)
6:     if $n = 1$ then
7:         break
8:     end if
9:     for all $p \in \mathcal{P}$ do
10:        $\mathcal{P}' \leftarrow \{ldcf(p), disc(p)\} \cup \{res(p,q) \mid q \in \mathcal{P}_n\}$
11:        $\mathcal{P}_{n-1} \leftarrow \mathcal{P}_{n-1} \cup \mathcal{P}'$
12:        COMPLETE_PROJECTION($\mathcal{P}'$, $n-1$)
13:    end for
14: end function
```

SMT-RAT consists of several modules that can be composed to an SMT solver. Each module is an SMT-compliant implementation of some decision procedure. SMT-compliancy includes three properties, see [9]:

1. incrementality: previous results should be reused and not recalculated if needed again
2. support backtracking according to the SAT solving
3. finding an infeasible subset as explanation for inconsistent constraint sets

It is possible to use some modules individually as theory solvers. But in the following SMT-RAT is regarded in its standard configuration as an SMT solver. It is a less lazy SMT solver, as described in the previous section, that uses the SAT solver MiniSAT, based on the DPLL algorithm, as SAT engine. The solver supports several optimizations such as conflict-driven clause-learning and theory propagation. Furthermore is SMT-RAT an incremental solver, which is one reason why the property SMT-compliancy is required for the modules.

One important part of SMT-RAT is the CAD module. In this module several different projection operators are implemented, amongst others McCallum's and Brown's operators. However in SMT-RAT the restriction of Brown's projection operator to be applied to a square-free basis is ignored. Therefore the restriction of McCallum's and Brown's operator is ignored in the CAD implementation. It does also not fix the projection if a polynomial vanishes during lifting. The impact of this was examined in [21]. It was found that McCallum's operator indeed produces some incomplete projections, but they did not cause an error on the benchmark set provided by the SMT-LIB.

In the CAD module are furthermore some optimizations implemented. For example constants as well as positive and negative definite functions are removed and not projected, since they are always sign- and order-invariant. In order to be SMT-compliant the CAD implementation also needs to be incremental. This means that the projection is implemented incrementally such that polynomials can be added and removed individually one after another. This is realized by projecting polynomials, instead of polynomial sets, in a depth-first manner, as in the simplified Algorithm 3, where any optimizations are ignored. The CAD itself is modified accordingly in an incremental lifting phase. How removing polynomials is implemented is explained in more detail in Section 4.2.

The advantage of an incremental projection is the reduced computational effort compared to a non-incremental implementation. This results from the less lazy SMT solving approach and the DPLL algorithm. The SAT solver searches for a solution for the Boolean abstraction of the input problem and if a conflict occurs or an intermediate solution is not feasible backtracking is applied. That way the SAT solver adds and removes a few constraints while usually most constraints remain the same. Therefore many similar CAD computations are needed, which means that large parts of previously computed CADs can be reused instead of recomputed with the incremental implementation.

Chapter 3

Restricted projection

In Section 2.2.1 Brown's projection operator is introduced. It is also explained that in the course of computing a projection using Brown's operator many discriminants and resultants need to be calculated. This is one of the reason why the CAD method is computationally expensive. The worst case complexity is doubly exponential in the number of both constraints and variables. In many cases a CAD can still be feasibly computed, but this method is nonetheless the most computationally expensive part of SMT-RAT, when SMT-RAT is applied to non-linear real algebra problems. Therefore it is desirable to reduce the computational complexity of the CAD module in practice.

There were already several improvements made since the CAD method was first introduced in [2] with Collins' original projection operator. A major progress was made by McCallum, he presented an advanced projection operator in [14], which is a subset of Collins' operator. This means less projection polynomials and therefore fewer cells in the CAD have to be computed, which results in a faster computation time. Unfortunately this new operator is only applicable for well-oriented polynomials. Additional improvements to this operator have later been suggested by Brown, as described in Section 2.2.1.

Definition 3.0.1 (Equational constraint).
A constraint of the form $p = 0$, for $p \in \mathbb{R}[x_1,...,x_n]$ is called equational constraint, p is called equational constraint polynomial.

Collins was the first to suggest a further improvement using equational constraints (Definition 3.0.1), see [8], while McCallum developed the theory behind this and gave a formal definition of the operator in [16]. The idea behind said improvement is that usually when constructing a CAD one is not interested in the signs of polynomials but in the truth values of the corresponding constraints. This means that a truth-invariant CAD as in Definition 3.0.2 is sufficient. A sign-invariant CAD for the defining polynomials is trivially truth-invariant for the corresponding constraints. However the other direction does not hold, a truth-invariant CAD may contain less cells than a sign-invariant one.

Definition 3.0.2 (Truth-invariance).
A CAD is called truth-invariant if the truth value of each constraint is constant on each cell of the CAD.

The concrete approach to reduce the complexity, suggested by McCallum, is to reduce the number of resultants that need to be computed by using equational con-

Algorithm 4 The EC-restricted CAD algorithm [12]

```
Input: polynomial $e \in \mathbb{R}[x_1,...,x_n]$, set of polynomials $P \subset \mathbb{R}[x_1,...,x_n]$.
Output: CAD of $\mathbb{R}^n$ invariant with respect to an $e$ or FAIL if the input is not
    well-oriented.
 1: $E \leftarrow \{e\}$
 2: Compute the finest square-free basis $F$ of $prim(E)$.
 3: if $n = 1$ then
 4:     return CAD on $\mathbb{R}$ formed by the decomposition of $\mathbb{R}$ according to the real
    roots of the polynomials in $F$
 5: end if
 6: $A \leftarrow P \cup E$
 7: $C \leftarrow cont(A)$
 8: Compute the finest square-free basis $B$ for $prim(A)$.
 9: $\mathcal{P} \leftarrow C \cup Proj_F(B)$
10: Compute the other projection levels using McCallum's projection operator and
    construct the CAD.
```

straints. Therefore he defines a restricted projection operator $Proj_E$, see Definition 3.0.3 [16]. This operator is applied instead of the McCallum operator in the first projection level only, then the projection phase is continued with the original McCallum operator, see Algorithm 4. His approach can be applied in combination with projection operators other than his own as well.

Definition 3.0.3 (Restricted projection operator).
For a set of integral polynomials $A \subset \mathbb{R}[x_1,...,x_n]$ and $E = \{e\} \subset A$, for an equational constraint polynomial e, the restricted projection of A relative to E is

$$\begin{aligned} Proj_E(A) &= cont(A) \cup Proj(F) \cup \{res(e,g) \mid e \in F, g \in B, g \notin F\} \\ &= \{lcf(e),\ disc(e) \mid e \in F\} \cup \{res(e,g) \mid e \in F, g \in B, g \notin F\} \\ &\subset \mathbb{R}[x_1,...,x_{n-1}] \end{aligned}$$

with the finest square-free basis B for $prim(A)$ and the finest square-free basis F for $prim(E)$.

The result when applying $Proj_E$ is not a sign-invariant CAD but a CAD that is sign-invariant with respect to $e = 0$ and sign-invariant with respect to the other constraints in those cells where $e = 0$. Such a CAD is called invariant with respect to an equational constraint. It could also be understood as a truth-invariant CAD for the equational constraint and a sign-invariant one for the other polynomials when the equational constraint is satisfied.

$Proj_E$ might be applied in subsequent levels as well, however this is not yet proven. McCallum gave a proof that validates the use of P_E in the first projection level in [16], as well as in both levels for a $3-$*dimensional* CAD. Further examinations are presented in [17]. A complexity analysis can be found in [6].

So far it was only discussed how this operator improves the projection phase by creating less polynomials though the lifting phase can also be optimized, which is described in [12] but not part of this thesis. It should also be considered that the savings due to the improved projection operator become already magnified throughout the rest of the algorithm without optimizing the lifting, which is especially important for problems with a higher number of variables.

In [6] McCallum's reduced projection operator is generalized for a truth-invariant CAD for a set of quantifier-free formulas. But since the implementation of an improved projection operator in the course of this thesis is supposed to be part of the CAD module in SMT-RAT this is not of interest here, because in SMT-RAT the CAD method is used as a theory solver that only solves conjunctions of constraints. Another approach that should be mentioned in this thesis is the semi-restricted projection.

3.1 Semi-restricted projection

The semi-restricted projection operator is presented in [17]. It makes use of the resultant rule in Theorem 3.1.1, which propagates equational constraints. This way more polynomials in the projection set are classified as equational constraint polynomials and can be used for semi-restricted projection.

Theorem 3.1.1 (Resultant rule).
If e_1 and e_2 are both equational constraint polynomials their resultant $res(e_1,e_2)$ is a propagated equational constraint polynomial, since $e_1 = 0 \wedge e_2 = 0 \Rightarrow res(e_1,e_2) = 0$.

For the semi-restricted projection operator defined in Definition 3.1 the repeated application is valid, wherever it is applicable throughout the projection phase. It is shown that the restricted projection operator $Proj_E$ can be used for the first and last projections, and the semi-restricted projection operator $Proj_E^*$ for every other projection. This allows to use equational constraints (propagated or present in the input) at every projection level to reduce the size of the projection sets. Since P_E can only be applied in the first and last projections the reduction in size is greatest in these levels.

Definition 3.1.1 (Semi-restricted projection operator).
For a set of pairwise relatively prime irreducible integral polynomials $A \subset \mathbb{R}[x_1,...,x_n]$ and $E = \{e\} \subset A$, for an equational constraint polynomial e, the semi-restricted projection of A relative to E is

$$\begin{aligned} Proj_E^*(A) &= Proj_E(A) \cup \{disc(g) \mid g \in A, g \notin E\} \\ &\subset \mathbb{R}[x_1,...,x_{n-1}]. \end{aligned}$$

For a set of integral polynomials $A \subset \mathbb{R}[x_1,...,x_n]$ and $E = \{e\} \subset A$, for an equational constraint polynomial e, the restricted projection of A relative to E is

$$\begin{aligned} Proj_E^*(A) &= cont(A) \cup Proj_F^*(B) \\ &\subset \mathbb{R}[x_1,...,x_{n-1}] \end{aligned}$$

with the finest square-free basis B for $prim(A)$ and the finest square-free basis F for $prim(E)$.

Now that the theory of restricted projection using equational constraints is introduced the next chapter deals with the implementation of this projection in SMT-RAT.

Chapter 4

Implementation

In this chapter the extensions of the CAD module in SMT-RAT (introduced in Section 2.3.1) that were implemented in the context of this paper are described. The first extension are a slightly modified data structure to store which polynomials are bounds or purged by bounds and more regular checks whether polynomials can be neglected due to bounds. Secondly are polynomials that are no longer needed, for example due to the removal of a constraint, deactivated instead of deleted, which aims to make better use of the incrementality of the module. And lastly is the implementation of the concept presented in Chapter 3 explained.

4.1 Simplification using bounds

Bounds (see Definition 4.1.1) are constraints that contain linear, univariate, polynomials. This special type of constraints can be used to simplify the projection, which is illustrated in Example 4.1.1. Some polynomials that contain the variable that is restricted by a bound may be neglected, more precisely those that are for all permitted values of said variable either always positive or always negative.

Definition 4.1.1 (Bounds).
A constraint of the form $x + a \Diamond 0$*, with* $a \in \mathbb{R}$*,* $\Diamond \in \{<, \leqslant, =, \neq, \geqslant, >\}$ *and a variable* x *is called a bound.*

Example 4.1.1 (Simplification due to bounds).
Let a projection contain amongst others the polynomials $p_1 = x^3 \cdot y^2 + 1$*,* $p_2 = x^2 + y$ *and* $p_3 = y^4 \cdot z^3$*. May the bound* $x > 0$ *become added to the projection. Then the polynomials are evaluated as follows:*

1. p_1 *is positive for all values for* x *such that the bound is satisfied, despite the occurrence of* y*, therefore* p_1 *can be neglected*
2. p_2 *contains* x *but its sign is also depending on* y*, the polynomial has still roots*
3. p_3 *does not contain* x *therefore the bound has no impact on the possible values of* p_3

When a polynomial can be neglected due to the bounds no successors (leading coefficient, discriminant and resultants) need to be computed for this polynomial. Since the computation of the discriminants and resultants is one of the main computational efforts in the CAD method, it can save some computation time to evaluate which polynomials are purged since for these and their successors respectively no discriminants and resultants need to be computed. Therefore this check is performed whenever a bound is added to the projection. It also has to be tested which polynomials can no longer be neglected when a bound is removed. Due to the fact that in the current implementation the adding of new polynomials and the computation of the projection are separate methods these checks could also be performed before computing the projection in case one or several bounds were added or removed before. However for the better comprehensibility of the implementation and considering that this check can be performed relatively quickly here this is checked directly when adding or removing a bound. It could be tested whether in practice the check is executed less often with the alternatively suggested approach however it is not expected to cause a significant reduction of the total running time.

Example 4.1.2 (Relevant levels for purging)**.**

$$
\begin{array}{lll}
\mathcal{P}_n & p_1 & = x_i^3 \cdot x_n^2 + 1, \ldots \\
\mathcal{P}_{n-1} & p_2 & = x_{n-1} + x_i^2,\ p_3 \ = \ x_{n-1} + x_{n-3}^2, \ldots \\
\vdots & \vdots & \\
\mathcal{P}_i & p_4 & = x_i^2 \cdot x_{i-2} - 1,\ p_5 \ = \ x_i^3,\ x_i > 0, \ldots \\
\vdots & \vdots & \\
\mathcal{P}_1 & p_6 & = x_1^3, \ldots
\end{array}
$$

The check which polynomials can be neglected is limited to the relevant polynomials, illustrated in Example 4.1.2. Since a bound for the variable x has no impact on polynomials in projection levels where x was already eliminated only levels where x occurs are checked. Furthermore we distinguish between adding and removing a bound. When adding a bound polynomials that are already neglected due to other bounds (in the example gray) do not have to be checked again. Respectively when removing a bound only polynomials that were neglected before may become relevant again. The polynomials that are due to bounds inactive are marked in a vector of bitsets and not deleted, because this would be impractical when a bound is removed and the polynomials become necessary again.

4.2 Inactive polynomials

In the section before is described how polynomials become deactivated if they can be neglected due to bounds. Another reason for polynomials to become unneeded is the removal of a constraint. It is desirable to deactivate polynomials in this case as well, instead of removing them, in case the constraint is added again later in the decision process. How this is handled is explained in this section.

When a constraint is removed all polynomials resulting from the corresponding polynomial become irrelevant. This includes not only direct successors of the polynomial but also their successors recursively. Formerly all these polynomials were deleted. This is quite inconvenient if the constraint is added again later, because then the polynomials have to be recalculated. The case that the necessity of polynomials

Algorithm 5 Active polynomial

```
function isActive(polynomial $p$)
    if isPurged($p$) then                          ▷ $true$ if $p$ is neglected due to bounds
        return $false$;
    else
        for all origin $o$ of $p$ do
            if $o.active_1 \wedge o.active_2$ then
                return $true$;
            end if
        end for
        return $false$;
    end if
end function
```

changes occurs even more often when the restricted projection described in Chapter 3 is used, and we see more about this in the next section.

These circumstances require to keep polynomials even if they are currently not part of the projection in order to save computation time. Therefore instead of removing polynomials they become deactivated. In the section before deactivation could be achieved by simply setting a bit corresponding to the polynomial. This is not practicable here, especially later when it comes to implementing the restricted projection. The reason for this is that there may be several different causes for a polynomial to be inactive, as indicated by Example 4.2.1. To update whether a polynomial is active would require to check all predecessors in the projection, which would be quite time consuming due to the size of the projection. Therefore are not just the direct predecessors stored for each polynomial but also for each tuple of predecessors the reasons for inactivity.

Example 4.2.1 (Multiple predecessors).
Let a projection contain amongst others the polynomials p, p_1, p_2, p_3 and p_4. Furthermore let $p = res(p_1,p_2) = res(p_3,p_4)$.

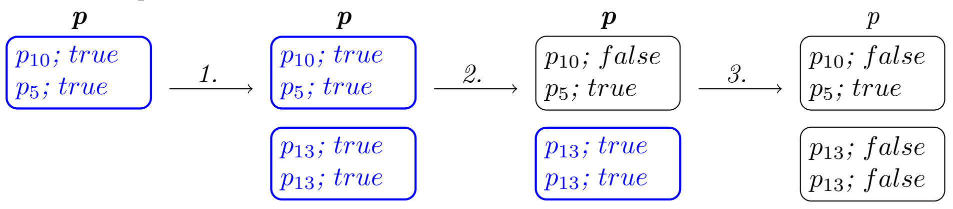
- *Assume p_1 is removed. Then p is still part of the projection due to $p = res(p_3,p_4)$.*
- *Assume p_3 is removed as well, then p is supposed to be inactive.*

In the implementation this is realized as follows. For each polynomial p a list of origins is stored. Every origin (see Definition 4.2.1) contains a level and two polynomials in this level (the direct predecessors of p). Now the origins store additionally two Booleans, one for each polynomial, which indicate whether the polynomials are active. If a predecessor polynomial is inactive the corresponding Boolean is $false$.

Definition 4.2.1 (Origin data structure).
polynomial predecessor$_1$; bool active$_1$
polynomial predecessor$_2$; bool active$_2$

Then the list of origins of a polynomial can be used to determine whether it is active, which is shown in Algorithm 5. Each origin is active if both predecessors are active, otherwise inactive. The polynomial is then active if it has at least one active origin and is not neglected due to bounds (see Section 4.1).

Example 4.2.2 (BaseType)**.**
1. Add p_{13} with $p = disc(p_{13}, p_{13})$.
2. Remove p_{10}.
3. Remove p_{13}.



This is illustrated without bounds in Example 4.2.2. There blue is used for active BaseTypes and black for the inactive ones. An active polynomial is denoted with **p**, while p is inactive.

4.3 Use of equational constraints

After describing how to deactivate polynomials in general in the last section here the implementation of the restricted projection as defined in Chapter 3 can be introduced. It is started with a description of what exactly is implemented followed by an explanation about how it is implemented.

In principle the restricted projection operator as in Definition 3 is implemented, however, as mentioned in Subsection 2.3.1 does SMT-RAT not calculate a square-free basis. This restriction of the projection operator is therefore ignored. It is also mentioned before that the use of the restricted projection operator is not yet validated for more than just the first projection level. Nevertheless is the restricted projection operator in this implementation used in as many directly successive levels as possible, starting with the first level.

When implementing this restricted projection it has to be ensured that the CAD module remains an SMT-compliant theory solver. SMT-compliancy is defined in Subsection 2.3.1 and includes incrementality. This means it has to be possible to add and remove individual polynomials, including *equational constraint polynomials (ECPs)*. Therefore the formerly used function to add polynomials described simplified in Algorithm 3 needed to be modified. The new (still simplified) version of this function can be found in Algorithm 6. The input EC is $true$ if p is an ECP.

Definition 4.3.1 (Conditions for restricted projection)**.**
No other equational constraint polynomial is used for restricted projection in lvl yet and $lvl = n$ or in $lvl - 1$ another equational constraint polynomial is used for restricted projection already.

The main difference to the former version is the execution of the method RESTRICT_PROJECTION. In this method nothing happens if EC is $false$. But when adding a new ECP it has to be checked whether it can be used to apply the restricted projection operator instead of Brown's operator. The conditions under which the restricted projection operator can be used with the new ECP added to the projection level lvl are listed in Definition 4.3.1, it is assumed that level n is the first level. All three conditions need to hold. If the ECP can not be used the method does nothing but this check.

Algorithm 6 Adding polynomials in an incremental projection considering ECPs

```
1: function ADD_POLYNOMIAL(polynomial $p$, $EC$)
2:     $level$ ← MAIN_VARIABLE($p$)                    ▷ index of $p$'s main variable
3:     $\mathcal{P}_{level} \leftarrow \mathcal{P}_{level} \cup \{p\}$
4:     RESTRICT_PROJECTION($p$,$EC$,$level$)
5:     COMPLETE_PROJECTION($\{p\}$,$level$)
6: end function

7: function COMPLETE_PROJECTION(set of polynomials $\mathcal{P}$, index of main
   variable $level$)
8:     if $level = 1$ then
9:         break
10:    end if
11:    for all $p \in \mathcal{P}$ do
12:        if USE_EC($level$) then      ▷ $true$ if restricted projection is used in $level$
13:            $\mathcal{P}' \leftarrow \{res(p,e)\}$                   ▷ for the used ECP $e$
14:        else
15:            $\mathcal{P}' \leftarrow \{ldcf(p), disc(p)\} \cup \{res(p,q) \mid q \in \mathcal{P}_{level}\}$
16:        end if
17:        $\mathcal{P}_{level-1} \leftarrow \mathcal{P}_{level-1} \cup \mathcal{P}'$
18:        COMPLETE_PROJECTION($\mathcal{P}'$, $level - 1$)
19:    end for
20: end function
```

It should be noted that to check these conditions for each level has to be stored whether an ECP is used for restricted projection in that level and if so which one. Therefore these information are stored if the new ECP turns out to be usable for restricted projection. In that case are furthermore all polynomials that become negligible deactivated, how exactly is described later. These are basically all polynomials in the next projection level and all their successors, since only the leading coefficient and the discriminant of the ECP as well as the resultants of each polynomial in *lvl* and the ECP are needed in the next level. An exception are only polynomials that were directly added to the next level from the input and their successors.

When the restricted projection operator could be applied for *lvl* it is checked whether it can also be applied for the next level, since there might be another ECP that could not be used for restricted projection before due to the absence of an ECP in a higher level. In that case the above described process is executed repeatedly until a projection level with no ECP occurs.

Additionally to the previously described change COMPLETE_PROJECTION is slightly modified. In case a polynomial p is added to a level in which the restricted projection operator is used only the resultant of p and the used ECP is added to the next level. Otherwise the method behaves the same as before.

The method to remove polynomials had to be changed as well, just like the method to add polynomials described above. The changes basically reverse those in ADD_POLYNOMIAL. If an ordinary polynomial is removed the method behaves the same as before and just deactivates the polynomial and all its successors. In case an ECP should be removed there are several possibilities. Either the ECP is not used for restricted projection, then it is removed like any other polynomial, or the ECP is used for restricted projection. In the latter case there are again two possibilities. There might be another ECP in the same level which can be used for restricted projection instead, then this is done and the projection is changed suitably. If there is no other ECP the projection operator needs to be changed back to Brown's operator and the projection must be expanded accordingly. This has to be done for all subsequent levels, which are using the restricted projection operator, as well.

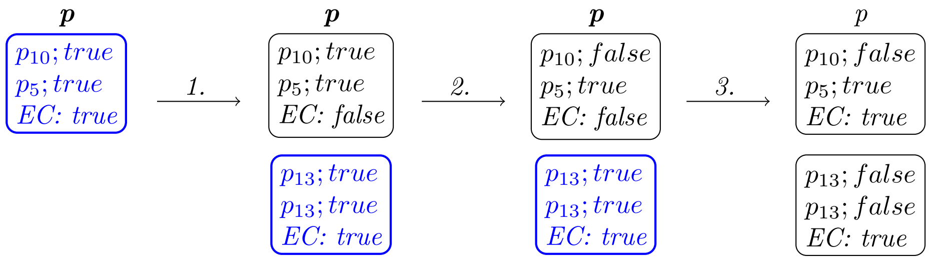
In the next subsection is briefly explained how polynomials become deactivated in this implementation of the restricted projection.

4.3.1 Deactivation

To deactivate polynomials due to an ECP used for restricted projection the BaseTypes mentioned before in Section 4.2 are extended by another Boolean that indicates whether the BaseType is inactive due to an ECP. This Boolean is set to *true* in a BaseType (lvl,id_1,id_2), if and only if in the projection level lvl an ECP with ID id_e is used for restricted projection and $id_1 \neq id_e$ and $id_1 \neq id_e$ hold.

An extended BaseType is active iff none of the additional Booleans is *true*. And a polynomial p is active exactly when at least one BaseType in its origin is active and p is not purged by bounds.

Example 4.3.1 (Extended BaseType)**.**
1. Add ECP p_{13}*.*
2. Remove p_{10}*.*
3. Remove p_{13}*.*



This concept is illustrated in Example 4.3.1. It should be noted that for successors of a polynomial p, which is inactive due to an ECP, the reason for inactivity is the inactivity of p and not an ECP, unless there is one in the same level as p used for restricted projection.

Chapter 5

Experimental results

In this chapter the implementations described in Chapter 4 are examined as part of the CAD module in SMT-RAT. Therefore a set of input problems is necessary. Since SMT-RAT accepts the problem description format .smt2, which is a common standard specified in the SMT-LIB [3], as input format the set of QF_NRA benchmarks also provided by the SMT-LIB is used. In the following this set of benchmarks is called QF_NRA.

To analyze the performance of the implementations made in the course of this thesis, SMT-RAT is executed on the problems in QF_NRA with different settings and a time limit of 30 seconds. The possible outputs are the following:

- *sat* and *unsat* - the problem was solved correctly and is satisfiable respectively unsatisfiable
- *timeout* - the solver did not find a solution within 30 seconds
- *memout* - the solver used all the available memory before finding a solution
- *wrong* - the solver computed a result but it was not the correct solution

First it has to be checked that no *wrong*s occur, when executing SMT-RAT while using the restricted projection, since in this implementation the operator might be used repeatedly, which is not validated. On this set of benchmarks the solver works correctly, despite using the restricted operator not just in the first level. With that being ensured the implementation can be examined.

Five different settings are regarded: A default setting **Default**, where the former implementation is used, to test whether any of the changes improved the previous implementation and four settings for different combinations of the new implementations. These four settings all include to deactivate polynomials instead of deleting them, as described in Section 4.2. The first one has no other difference to **Default** and is called **InAct**. In the second one bounds are used to simplify the projection (see Section 4.1), this one is called **B**, while in the third, named **EC**, the restricted projection operator explained in Section 4.3 is used. And finally the forth setting includes both and is called **EC_B**.

The overall solver performance for these five settings is shown in Table 5.1. The columns SAT and UNSAT contain the numbers of examples that were correctly solved with the respective solution. And in the column no result are the problems with outputs *timeout* and *memout* collected. It can be seen that with the **Default** setting

Table 5.1: Overall solver performances

Settings	SAT	UNSAT	no result
Default	4852	3905	3327
InAct	4577	3778	3729
B	4627	3770	3687
EC	4559	3782	3743
EC_B	4609	3769	3706

approximately 260 more problems that are satisfiable and 130 more unsatisfiable problems are solved than with the other settings. Amongst the four settings using the new implementation the differences are rather small. These results are unforeseen because the changes were expected to reduce the computational effort and the complexity of the projection and therefore enable SMT-RAT to solve more problems, but this is apparently not the case. So next it is necessary to investigate why the overall performance is worse.

One reason for the worse results might be the increased memory consumption due to the fact that polynomials are just deactivated and not removed, which might make a difference for large problems. There are more *memout*s with **InAct**, namely ≈ 380 compared to ≈ 180 with **EC** involved and ≈ 110 for Default.

Table 5.2: Complexity for different settings

Settings	projection size	computed polynomials	avg. running time (ms) on solved
Default	6.690	9.222	555.37
InAct	4.922	7.124	623.35
B	6.438	4.960	599.71
EC	4.709	6.465	606.45
EC_B	6.281	4.427	585.45

For the more detailed examination in Table 5.2 only the 8247 problems that could be solved under all settings are considered. The column projection size contains the average of the maximum projection sizes for each problem, while the column computed polynomials contains the average number of total computed polynomials, which are leading coefficients, discriminants and resultants. Both projection size and number of computed polynomials decreased for the settings including deactivation as expected. Still the average running times are worse than when using **Default**.

Amongst the four settings with deactivation **InAct** is on average the slowest, so compared to this basic setting the simplification by using bounds and the restricted projection do speed up the computations. The best average running time occurred for the setting that combined both of them. The average numbers of computed polynomials fit this observation as well. However the projection size does not, since it is smaller when bounds are not used for simplification.

When using the settings **B** or **EC_B** the average number of purged polynomials per problem is 0.027, respectively 0.015, which corresponds to on average one polynomial in every 37th respectively 67th problem. Besides was at least one ECP added to the projection in 2752 problems and with the settings **EC** and **EC_B** was the restricted projection operator used at least once in 1645 of these problems. From these numbers in combination with the results in Table 5.2 it can be concluded that the implemented changes are actually applied to some problems and they also do have positive effects on the running time and the number of computed polynomials, compared to **InAct**.

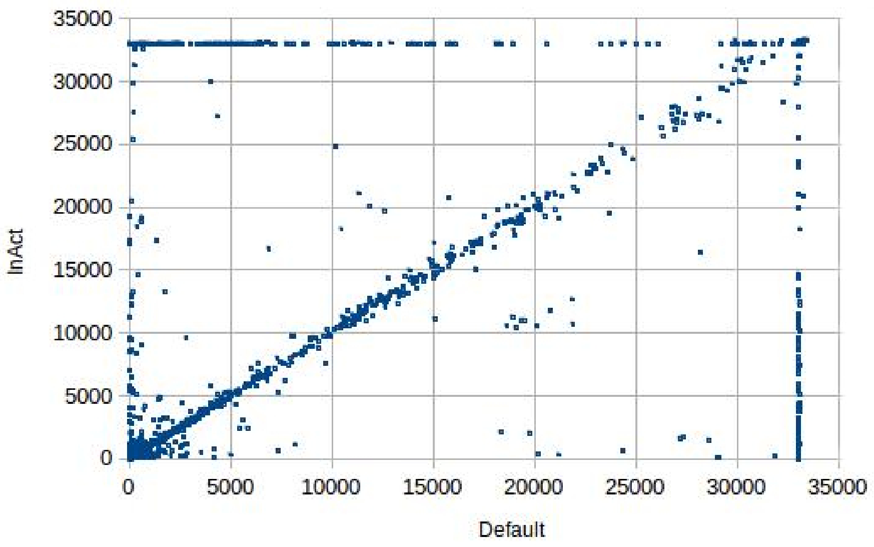


Figure 5.1: Running times of individual problems using **Default** and **InAct**

However **Default** has still better average running times, despite computing more polynomials. To find a reason for this first the running times of **Default** and **InAct** are compared for the individual problems. This is shown in Figure 5.1. The figure contains one point for each of the considered problems, which has the running time in milliseconds for **Default** on the problem as x-value and the one for **InAct** as y-value. The points on the diagonal represent problems for which with both settings similar running times are achieved. It stands out that there are many points on the lines parallel to the x- and y-axis through about 33000, which represent problems that could only be solved for one of the settings, while only a few points are in-between these three lines. A similar plot is achieved when comparing **Default** and **B**, see Figure 5.2.

The reason for this might be the incremental implementation, which is not entirely explained in this thesis. Incrementality in the sense of adding and removing polynomials individually is explained in Subsection 2.3.1, but it was not mentioned before in the sense of computing CADs incrementally when new polynomials are added. This means basically that partially computed CADs are used to check for satisfiability before the full projection is computed. If different polynomials are included in partial projections for different settings one of the projections might be sufficient to find a sample point to satisfy the constraints while the other is not. So even small differences in the projection seem to cause a different behavior.

This is also the case when using two settings that both include deactivation of polynomials, as can be seen in Figure 5.3. In this figure something else is noticeable: there are more problems that can be solved by using **B** but not **EC** than the other way

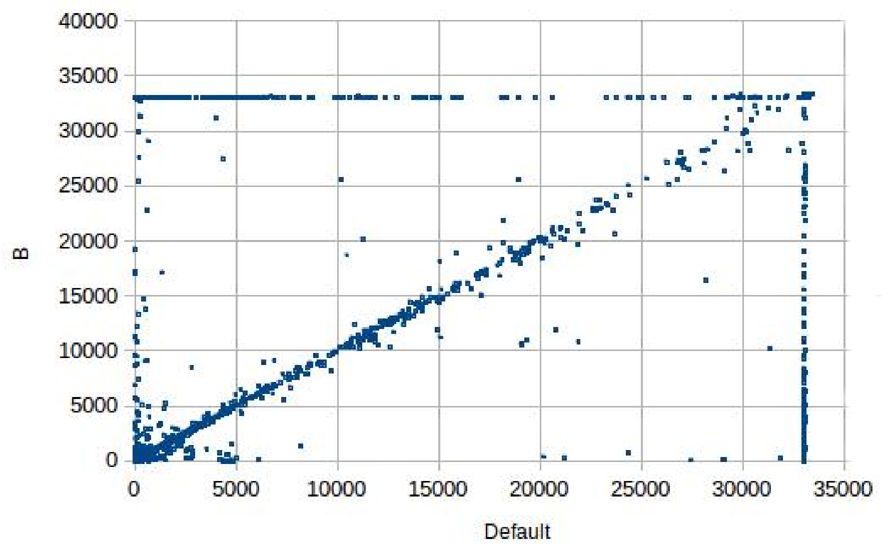


Figure 5.2: Running times of individual problems using **Default** and **B**

round. This might be explained by the degrees of the polynomials in the projection. The degree shrinks on the leading coefficients, while the degrees of discriminants and resultants are likely to grow. So a partial CAD based on leading coefficients results in less sample points and can return answers faster, but when using restricted projection with **EC** or **EC_B** less leading coefficients are added to the projection.

When it comes to lifting it should also be considered that, while polynomials are only deactivated and not anymore deleted, the corresponding elements in the CAD are currently deleted. However, a test showed that when these elements are neither deleted nor deactivated nearly no impact on the results (runtimes and solved benchmarks) is noticeable.

For further insight into the reasons why the implemented changes in the CAD method do neither improve the number of solved problems nor the average running time, additional statistics need to be collected and evaluated.

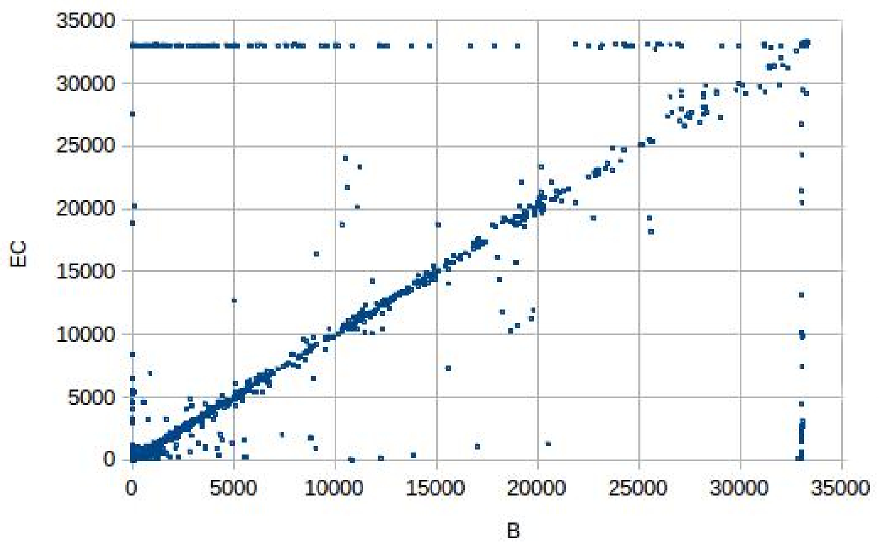


Figure 5.3: Running times of individual problems using **B** and **EC**

Chapter 6

Conclusion

This thesis started with an introduction to different decision problems and selected decision procedures, namely the DPLL algorithm for SAT and the CAD method for real algebra. It was continued with a description of the SMT solver SMT-RAT in which both methods are combined. Next the restricted projection operator by McCallum was explained. This operator allows to use equational constraints to compute less polynomials in the projection phase of the CAD method.

In the further proceeding is a description of the extension of the CAD method in SMT-RAT with an implementation of this restricted projection operator given. Furthermore is described how a simplification of the projection due to bounds and the deactivation of polynomials were implemented.

It turned out that the changes in the implementation did not have the desired effect to improve the overall performance of SMT-RAT, even though it did not cause any errors to use the restricted projection operator repeatedly. The projection size and the number of computed polynomials decreased when applying the modified method, however the number of solved problems and the average running time were worse. To gain further insight to the reasons why this is the case more statistics should be collected and evaluated.

It should be noted that using the restricted projection operator did improve the average running time compared to a setting where only the deactivation of polynomials was used. Applying the simplification due to bounds improved the result even further, so did a combination of both. It could be tried to implement these approaches for a CAD implementation where polynomials are deleted instead of deactivated. Further possible future work is suggested in the next section.

6.1 Future work

There are several options to continue this work, some are already mentioned in the previous chapters. One possibility is to change the implementation for the lifting phase as well to deactivate sample points instead of removing them. Currently the polynomials in the projection are deactivated and not removed, however the corresponding sample points are still removed. So when polynomials are often removed and again added several sample points might be computed repeatedly, which is time consuming as well. Another possibility for improvement is the variable order, which is discussed in the next subsection.

6.1.1 Variable ordering

A CAD depends heavily on the variable order. When using the restricted projection operator the variable order may have an even bigger impact on the CAD, particularly on the complexity, than with a conventional projection operator. This is illustrated in the Example 6.1.1. Therefore it could be tried to find a variable order to use the restricted projection operator in as many levels as possible.

Example 6.1.1 (Impact of the variable order).
Given the following set of polynomials $\mathcal{P}_n = \{e_3 = x_3 \cdot x_2 + x_1;\ e_2 = x_2^2 \cdot x_1;\ e_1 = 5 \cdot x_1^5;\ ...\}$*, where* e_1, e_2 *and* e_3 *are equational constraint polynomials, the size of the projection depends strongly on the variable order:*

$$\begin{array}{lll} & x_1 & \mathcal{P}_3 = \{e_3;\ e_2;\ e_1;\ ...\} \\ 1. & x_2 & \mathcal{P}_2 \\ & x_3 & \mathcal{P}_1 \end{array}$$

$$\begin{array}{lll} & x_3 & \mathcal{P}_3 = \{e_3;\ ...\} \\ 2. & x_2 & \mathcal{P}_2 = \{e_2;\ ...\} \\ & x_1 & \mathcal{P}_1 = \{e_1;\ ...\} \end{array}$$

In the Example 6.1.1 two variable orders are examined. In the first case the restricted projection operator can only be applied in the first projection level. Compared to this case the variable order in the second case seems advantageous, since in all three levels the restricted projection operator can be used. However it has to be considered that the CAD implementation is incremental. Lets assume that the constraint corresponding to the equational constraint e_3 becomes removed, see Example 6.1.2. In that case using the first variable order the restricted projection operator can still be applied, only for a different equational constraint polynomial, but when using the second variable order the restricted projection operator can not be applied anymore at all.

Example 6.1.2 (Variable order in an incremental projection).
New set of polynomials $\mathcal{P}_n = \{e_2 = x_2^2 \cdot x_1;\ e_1 = 5 \cdot x_1^5;\ ...\}$.

$$\begin{array}{lll} & x_1 & \mathcal{P}_3 = \{e_2;\ e_1;\ ...\} \\ 1. & x_2 & \mathcal{P}_2 \\ & x_3 & \mathcal{P}_1 \end{array}$$

$$\begin{array}{lll} & x_3 & \mathcal{P}_3 \\ 2. & x_2 & \mathcal{P}_2 = \{e_2;\ ...\} \\ & x_1 & \mathcal{P}_1 = \{e_1;\ ...\} \end{array}$$

Regarding this problem the incremental projection may cause when using the restricted projection operator, it might be a good approach to aim for a variable ordering like the first one in the above examples and additionally implement the resultant rule, which propagates ECPs such that restricted projection can be used more often. Another promising approach is to implement the semi-restricted projection mentioned in Subsection 3.1, which could be used in every projection level as long as its applicable and not just in consecutive ones.

Bibliography

[1] Erika Abraham and Gereon Kremer. Satisfiability checking: Theory and applications. In *Proceedings of the 14th International Conference on Software Engineering and Formal Methods (SEFM'16)*, volume 9763 of *LNCS*, pages 9–23. Springer International Publishing, 2016.

[2] Dennis S Arnon, George E Collins, and Scott McCallum. Cylindrical algebraic decomposition i: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.

[3] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[4] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. *Satisfiability modulo theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. 1 edition, 2009.

[5] S. Basu, R. Pollack, and M.F. Coste-Roy. *Algorithms in Real Algebraic Geometry.* Algorithms and Computation in Mathematics. Springer Berlin Heidelberg, 2013.

[6] Russell Bradford, James H. Davenport, Matthew England, Scott McCallum, and David Wilson. Truth table invariant cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 76:1 – 35, 2016.

[7] Christopher W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447 – 465, 2001.

[8] George E. Collins. Quantifier elimination by cylindrical algebraic decomposition - twenty years of progress. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 8–23, Vienna, 1998. Springer Vienna.

[9] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. Smt-rat: An open source c++ toolbox for strategic and parallel smt solving. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 360–368, Cham, 2015. Springer International Publishing.

[10] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[11] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[12] Matthew England. An implementation of CAD in maple utilising problem formulation, equational constraints and truth-table invariance. *CoRR*, abs/1306.3062, 2013.

[13] Hoon Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 166–173, Vienna, 1998. Springer Vienna.

[14] Scott McCallum. An improved projection operation for cylindrical algebraic decomposition, 04 1985.

[15] Scott McCallum. An improved projection operation for cylindrical algebraic decomposition of three-dimensional space. *Journal of Symbolic Computation*, 5(1):141 – 161, 1988.

[16] Scott McCallum. On projection in cad-based quantifier elimination with equational constraint. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, ISSAC '99, pages 145–149, New York, NY, USA, 1999. ACM.

[17] Scott McCallum. On propagation of equational constraints in cad-based quantifier elimination. In *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, ISSAC '01, pages 223–231, New York, NY, USA, 2001. ACM.

[18] Roberto Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[19] A. Tarski. *A decision method for elementary algebra and geometry.* Rand report. Rand Corporation, 1948.

[20] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.

[21] Tarik Viehmann, Gereon Kremer, and Erika Ábrahám. Comparing different projection operators in cylindrical algebraic decomposition for smt solving.

[22] Christopher W. Brown. The McCallum projection, lifting, and order-invariance. 09 2001.