Evaluation of Equational Constraints for CAD in SMT Solving

Rebecca Haehn, Gereon Kremer, and Erika Ábrahám

RWTH Aachen University, Germany

Abstract. The cylindrical algebraic decomposition algorithm is a quantifier elimination method for real-algebraic formulae. We use it as a theory solver in the context of satisfiability-modulo-theories (SMT) solving to solve sequences of related real-algebraic satisfiability questions.
In this paper, we consider some optimizations for handling equational constraints. We review some previously published ideas, in particular Brown's projection operator and some improvements suggested by McCallum. Then we discuss different variants of the restricted projection operator to implement them in our SMT solver SMT-RAT and provide experimental results on the SMT-LIB benchmark set QF_NRA. We show that the performance improves especially for unsatisfiable inputs.

1 Introduction

Solving the *satisfiability (SAT)* problem is an important sub-problem in many applications, ranging from program analysis [GAB+17] to industrial configuration management [Vol15] or the dependency management in Linux package managers [ADCTZ11]. An extension to the regular SAT problem is the *satisfiability-modulo-theories (SMT)* problem that allows for richer logics. While SAT only allows for propositional formulae, SMT deals with quantifier-free first-order logic formulae over one or more theories. One popular theory is the theory of *(nonlinear) real arithmetic*.

An SMT solver [BHvMW09,KS08] is traditionally separated into a SAT solving part – which makes use of a regular SAT solving engine – and a theory solving part. The SAT solving part deals with the logical structure of the formula and issues theory queries to the theory solving part. The *theory solving module* checks sets of theory constraints for consistency.

A number of theory solving modules for non-linear real arithmetic have been proposed, including incomplete algorithms like *interval constraint propagation* [GGI+10,HR97] or *virtual substitution* [Wei97]. In this paper, we deal with the *cylindrical algebraic decomposition (CAD)* method [Col75] which is, to the best of our knowledge, the only complete decision procedure for non-linear real arithmetic implemented in any SMT solver.

Given a set (conjunction) of real-algebraic input constraints and an ordering of the variables, the CAD method proceeds in two phases. It first uses a *projection operator* to produce a sequence of sets of polynomials of decreasing dimension.

The sets of real roots of these polynomials can be used to construct the borders of finitely many semi-algebraic sets that we call *cells*. The points in each of the cells are equivalent regarding the satisfaction of the input formula. In the second phase, samples are constructed dimensionwise for each of the cells. Thus to decide satisfiability it is sufficient to check whether any of the generated samples satisfies the formula.

A number of different projection operators exist, most notably the ones due to Collins [Col75], Hong [Hon90], Lazard [Laz94,MPP17], McCallum [McC85], and Brown [Bro01]. They use the same ingredients and mainly differ in the size of the sets of polynomials, essentially representing the continuing research in this area. It should be noted that the projection operators due to McCallum and Brown are incomplete in the sense that they may not generate some polynomials that are needed to find satisfying solutions. Past studies [VKÁ17] have shown, however, that this incompleteness does not surface on the SMT-LIB [BFT16] benchmarks.

In addition to the different projection operators, several modifications have been proposed to reduce the number of polynomials in certain cases. One particular modification that we analyse in this paper makes use of *equational constraints*. If equations are present in the input, they can be used to reduce the work for the projection.

In this paper, we present several possibilities to exploit the fundamental idea, in particular different variants of the restricted projection operator suggested by McCallum [McC01] in combination with Brown's projection operator [Bro01]. Then we discuss their impact on a CAD implementation used in the context of SMT solving in the SMT solver SMT-RAT and review some experimental results on the benchmark set QF_NRA provided by SMT-LIB. Some of the presented optimizations are proven to be sound while others might lead to incompleteness. However, similar to the projection operators that are incomplete in general, none of our optimizations yielded incorrect satisfiability results in our experiments.

After presenting some preliminaries in Section 2 we discuss the optimizations that we have implemented for the CAD projection phase in Section 3. In Section 4 we present and discuss our experimental results and conclude the paper in Section 5.

2 Preliminaries

2.1 SMT Solving

SMT solvers [BHvMW09,KS08] are used to decide the satisfiability of first-order logic formulae over one or more underlying theories. Traditionally, an SMT solver consists of a SAT solver and one or more theory solving modules. The SAT solver deals with the logical structure of the input formula by iteratively searching for solutions for its *Boolean skeleton*, which is the propositional logic formula obtained by replacing the theory constraints in the formula with fresh Boolean propositions. During this search, the theory solving modules are used to check

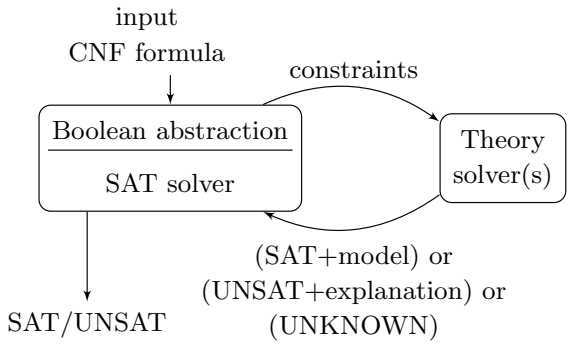

Fig. 1: The SMT solving framework [VKÁ17]

whether the current Boolean assignment is consistent with the theories. Therefore they check the consistency of a set of theory constraints, which contains all theory constraints whose corresponding proposition in the Boolean skeleton is assigned `true` and that appear non-negated in the formula, as well as those whose proposition is assigned `false` and that appear negated.

The above described so-called lazy SMT solving approach is illustrated in Figure 1. The frequency of theory checks varies in different approaches, in the *full* lazy approach they are only executed for full Boolean solutions, while in the *less* lazy approach they are already executed for partial solutions. If the theory constraints are conflicting then the theory solving module returns an explanation for the conflict and the SAT solver searches for a different Boolean solution informed by the explanation. In case the constraints are consistent in the theory, the Boolean assignment is either not yet complete, then the SAT solver continues its search; or it is complete, which means that a satisfying solution is found.

2.2 The CAD Method

The CAD method [Col75] can be used as a theory solving module for the theory of (*non-linear*) *real arithmetic*. It works with respect to a fixed, static variable ordering that we assume to be given.

Polynomials $p \in \mathbb{Z}[x_1, ..., x_n]$ with integer coefficients over variables $x_1, \ldots, x_n$ are expressions of the form $p = \sum_{i=1}^{m} a_i \prod_{j=1}^{n} x_j^{e_{ij}}$ with *coefficients* $a_i \in \mathbb{Z}$ and *exponents* $e_{i,j} \in \mathbb{N}_0$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The products $\prod_{j=1}^{n} x_j^{e_{ij}}$ are called *monomials*. If $n = 1$ then p is called *univariate*, otherwise *multivariate*. Multivariate polynomials in $\mathbb{Z}[x_1, ..., x_n]$ are usually interpreted as univariate polynomials in x_n with polynomial coefficients from $\mathbb{Z}[x_1, ..., x_{n-1}]$, i.e., as polynomials from $\mathbb{Z}[x_1, ..., x_{n-1}][x_n]$.

Polynomial *constraints* have the form $p \diamond 0$, where p is a polynomial in $\mathbb{Z}[x_1, \ldots, x_n]$ and where $\diamond$ is one of the comparison predicates $<$, $\leq$, $=$, $\neq$, $\geq$, $>$. The *sign* of a polynomial $p \in \mathbb{Z}[x_1, ..., x_n]$ in a point $r \in \mathbb{R}^n$ is defined as -1 if p evaluates in r to a strictly negative value, 1 for a strictly positive value, and 0 otherwise. To decide whether $p \diamond 0$ evaluates to `true` for a point r, it is sufficient to determine the sign of p under r, since p is compared to zero. If the sign of p is

the same under all points from a set $C \subseteq \mathbb{R}^n$ then C is called *p-sign-invariant*; in this case it suffices to determine the sign of p for a single point in the set to evaluate the constraint for all points in the set. Thus the satisfiability of p can be determined if we can construct a finite partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ of $\mathbb{R}^n$ into finitely many p-sign-invariant cells C_i, which we call a decomposition.

A decomposition $\mathcal{C} = \{C_1, \ldots, C_m\}$ is called *P_n-sign-invariant* for a set P_n of polynomials if it is p-sign-invariant for each polynomial from P_n. It is furthermore called *algebraic* if each $C_i \in \mathcal{C}$ is a connected semi-algebraic[1] set. Such a decomposition can be determined by the roots of the polynomials in P_n. It is additionally cylindrical when the cells are cylindrically ordered, which means that for each $C_i, C_j \in \mathcal{C}$ and each $1 \leq k < n$ the projections of C_i and C_j to $\mathbb{R}^{n-k}$ (by removing the last k coordinates) are either disjoint or identical.

For a set P_n of polynomials, a P_n-sign-invariant *cylindrical algebraic decomposition (CAD)* of $\mathbb{R}^n$ as described above can be computed using the CAD method. Though in general CAD works on real arithmetic formulae, in this paper we consider only sets of polynomial constraints as input, as we use the CAD method as an SMT theory solving module. The considered set P_n contains all polynomials of the input constraints.

The CAD method computes the CAD in two phases: the *projection phase* and the *lifting phase*. In the projection phase the polynomials that describe the boundaries of the CAD cells are computed using a *projection operator*. The projection operator is applied to the polynomials in $P_n \subseteq \mathbb{Z}[x_1, \ldots, x_n]$ to compute a set $P_{n-1} \subseteq \mathbb{Z}[x_1, \ldots, x_{n-1}]$ of polynomials, for which a CAD is computed recursively. The projection operator has the important property that each CAD $\mathcal{C}_{n-1}$ for P_{n-1} can be extended to a CAD $\mathcal{C}_n$ for P_n in an easy way by defining the cells of $\mathcal{C}_n$ to be the P_n-sign-invariant regions in the cylinders $C'_i \times \mathbb{R}$ for each cell $C'_i \in \mathcal{C}_{n-1}$.

There are several possible projection operators, in this paper, Brown's projection operator [Bro01] is used since compared to other operators it is faster and fewer polynomials have to be computed [VKÁ17]. In the following the properties *degree* $deg(p)$, and *leading coefficient* $lcf(p)$ of a polynomial p are used with the usual meaning. To define Brown's projection operator in addition the *Sylvester matrix* of univariate polynomials $p = \sum_{i=0}^{k} a_i x_n^i$ and $q = \sum_{i=0}^{l} b_i x_n^i$ in x_n with polynomial coefficients $a_i, b_i \in \mathbb{Z}[x_1, \ldots, x_{n-1}]$, $deg(p) = k \geq 1$, and

[1] A set $C \subseteq \mathbb{R}^n$ that can be described by a conjunction of polynomial constraints is called *semi-algebraic*.

$deg(q) = l \geq 1$, which is the following $(k+l)\times(k+l)$-matrix, needs to be defined.

$$Syl_{x_n}(p,q) := \begin{pmatrix} a_k & \cdots & a_0 & \cdots & 0 \\ & a_k & \cdots & a_0 & \vdots \\ \vdots & \ddots & & \ddots & \\ 0 & \cdots & a_k & \cdots & a_0 \\ b_l & \cdots & b_0 & \cdots & 0 \\ & b_l & \cdots & b_0 & \vdots \\ \vdots & \ddots & & \ddots & \\ 0 & \cdots & b_l & \ldots & b_0 \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} \text{l} \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} \text{k} \end{matrix}$$

Furthermore is the *resultant* of p and q defined as $res(p,q) = det(Syl_{x_n}(p,q))$, the *discriminant* of p as $disc(p) = det(Syl_{x_n}(p,p'))$, and the *content* of p as $cont(p) = gcd(a_0, ..., a_k)$, where *gcd* is the greatest common divisor. Finally Brown's projection operator $Proj$ itself is defined as follows: Let $P'_n = \{p'_1, \ldots, p'_m\}$ be the finest square-free basis of P_n.

$$\begin{aligned} Proj(P_n) = P_{n-1} = \{&lcf(p_i),\ disc(p_i),\ res(p_i,p_j) \mid p_i,\ p_j \in P'_n, i \neq j\} \\ &\cup \{cont(p_i) | p_i \in P_n \text{ and } cont(p_i) \text{ is non-zero, non-unit}\} \\ &\subset \mathbb{Z}[x_1, ..., x_{n-1}] \end{aligned}$$

Instead of polynomial sets, individual polynomials can be projected incrementally [CKJ+15]. The CAD itself is then modified accordingly in an also incremental lifting phase. This way the polynomials can be added and removed individually one after another. That enables to reuse large parts of previously computed CADs instead of recomputing them, which is useful since many similar CAD computations are needed when using the CAD method as a theory solving module in a less lazy SMT solver. This is due to the SAT solver that adds and removes only a few constraints while usually most constraints remain the same.

Given a P_n-sign-invariant CAD it suffices to consider one sample point from each cell to check whether any point in the cell satisfies the input constraints. These sample points are computed in the lifting phase. Since the CAD method is used to check whether a set of polynomial constraints is consistent, which is the case if we can assign a real value to each of the variables occurring in the set such that each constraint evaluates to true, it is also sufficient to compute only a partial CAD and stop the computation if a satisfying point is found [CH91].

3 Modification of the CAD Projection

When we construct a CAD for the purpose of theory solving in an SMT solver, we do not need the CAD to be *sign-invariant* on the input polynomials, instead we are content if every cell is invariant with respect to the evaluations of the constraints. We call a CAD *truth-invariant* if the truth value of every input constraint is constant on each cell.

Additionally, we can exploit the fact that we only use the CAD method as a theory solving module, which means that it is only used to check the consistency of *sets of constraints* instead or arbitrary Boolean combinations of constraints. This implies that a part of the solution space where a single constraint evaluates to `false` can be discarded as a whole, even if other constraints are not sign-invariant on this area. This *part of the solution space* may be more than a single cell of the CAD and we can try to avoid constructing individual cells within this part of the solution space altogether.

In order to exploit this, we consider modifications of the projection operators based on *equational constraints* [McC99]. These are all constraints of the form $p = 0$, where p is a polynomial as defined above, which we call an *equational constraint polynomial*, since we only consider conjunctions of constraints as input for the CAD method. Technically, McCallum suggested different ways to construct CADs that are sign-invariant with respect to equational constraints and sign-invariant with respect to the other constraints only in the cells where the equational constraints evaluate to `true`. Of course, this is only possible if equational constraints are present in the set of input constraints.

The advantage of these modifications is that we can use a coarser CAD using fewer polynomials in the projection phase that still answers our question. This CAD may consider fewer polynomials in the projection phase which leads to a smaller number of sample points in the lifting phase and eventually a fewer number of cells. Therefore we expect the modified method to scale better on larger inputs in the presence of equational constraints.

We note that we compute partial CADs, i.e., we let the CAD method terminate if a satisfying sample is found, and thus a full CAD is usually only computed on unsatisfiable inputs. We also observe that many satisfiable SMT problems do not produce a lot of unsatisfiable theory calls since most examples in the used benchmark set have little Boolean structure and large Boolean satisfying regions, therefore we expect the impact of this modification to be beneficial mainly on unsatisfiable inputs. Furthermore, the lifting phase may even profit if more polynomials are present that are comparably *easy* – for example with small degrees – in the projection as they may lead to a satisfying sample without considering *hard* polynomials at all. In such a case these modifications may actually hinder the lifting phase and slow down our solver.

3.1 Restricted Projection

The first modification suggested by McCallum is to restrict the projection operator that is applied in the first step of the projection phase [McC99]. Afterwards, we continue with the original projection operator.

McCallum suggested this approach for his projection operator, however, it can be applied in combination with projection operators other than his own as well. We use this restricted projection with the projection operator due to Brown [Bro01].

For a finite set of constraints with polynomials $P_n \subset \mathbb{Z}[x_1, ..., x_n]$ let $E \subseteq P_n$ a set of one polynomial which appears in an equational constraint and contains

the variable x_n that is to be eliminated first. The restricted projection of P_n relative to E is defined as follows:

$$\begin{aligned} Proj_E(P_n) = \quad & Proj(E') \cup \{res(e,p) \mid e \in E', p \in P'_n, p \notin E'\} \\ & \cup \{cont(p_i) \mid p_i \in P_n \text{ and } cont(p_i) \text{ is non-zero, non-unit}\} \\ = \quad & \{lcf(e),\ disc(e) \mid e \in E'\} \cup \{res(e,p) \mid e \in E', p \in P'_n, p \notin E'\} \\ & \cup \{cont(p_i) \mid p_i \in P_n \text{ and } cont(p_i) \text{ is non-zero, non-unit}\} \\ & \subset \mathbb{Z}[x_1, ..., x_{n-1}] \end{aligned}$$

with the *primitive part* of a set of polynomials P defined as $prim(P) = \{p/cont(p) \mid p \in P \text{ and } p/cont(p) \text{ is not constant}\}$ and the finest square-free basis P'_n for $prim(P_n)$ and the finest square-free basis E' for $prim(E)$. Note that we may only be able to apply this restricted projection operator under an appropriate variable ordering as x_n must be present in the equational constraint.

When using this restricted projection operator instead of the original projection operator less leading coefficients, discriminants, and resultants are added to the projection. This may reduce the size of the projection significantly which we hope to be beneficial for the overall performance. As mentioned before, the removal of comparably easy polynomials (in particular leading coefficients) may also be a disadvantage for satisfiable sets of constraints, though. We nevertheless hope a partial CAD using this modified projection operator to be faster on average.

The result when applying this approach is a CAD that is sign-invariant with respect to the used equational constraint and sign-invariant with respect to the other constraints in those cells where the equational constraint is satisfied. McCallum gave a proof that validates the use of $Proj_E$ in the first projection step in [McC99], as well as in both steps for a 3-dimensional CAD.

In our implementation, we apply $Proj_E$ on all levels – provided that appropriate equational constraints are part of the input. Though doing so is not formally proven to be sound, we base our application on the following argument: the places where the proof fails are statistically rare so in the context of solving many problems quickly we accept the risk.

3.2 Semi-Restricted Projection

Another modification suggested by McCallum is the semi-restricted projection operator [McC01], for which the repeated application is formally validated, wherever it is applicable throughout the projection phase. For sets of polynomials $P_n \subset \mathbb{Z}[x_1, ..., x_n]$ and $E = \{e\} \subset P_n$, where e is an equational constraint polynomial that contains the variable x_n, the semi-restricted projection of P_n relative to E is defined as follows:

$$\begin{aligned} Proj^*_E(P_n) &= Proj_E(P_n) \cup \{disc(p) | p \in P'_n, p \notin E'\} \\ &\subset \mathbb{Z}[x_1, ..., x_{n-1}] \end{aligned}$$

with the finest square-free basis P'_n for $prim(P_n)$ and the finest square-free basis E' for $prim(E)$.

McCallum has shown that this operator can be used whenever applicable and that alternatively the restricted projection operator $Proj_E$ can be used for the first step and the last step in the projection phase, and the semi-restricted projection operator $Proj^*_E$ in every other step [McC01]. For the latter combination a detailed complexity analysis can be found in [EBD15,ED16]. This allows using equational constraint polynomials at every projection step where one is present to reduce the size of the projection. Note that if the underlying projection operator is incomplete (like McCallum's or Brown's) then the restricted projection operator is also incomplete.

3.3 The Resultant Rule

McCallum proposed a method called the *resultant rule* [McC01] to exploit the semi-restricted projection even if no explicit equational constraint is present for a specific level. 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 $e_1 = 0 \land e_2 = 0 \Rightarrow res(e_1, e_2) = 0$. Due to this rule more polynomials in the projection are classified as equational constraint polynomials.

This method was only proposed for the semi-restricted projection, as the restricted projection was defined for the top level only. Given that we assume the restricted projection to be usable on all levels as well, we could also use the resultant rule when we apply the restricted projection multiple times. Currently, we do not use this rule in our implementation. The reasons are specific requirements for the underlying data structures, causing challenges for the implementation. Intuitively, a polynomial can be part of the projection due to several reasons. In the incremental setting, we would need to keep track of all possible reasons and apply postponed projection steps if e.g. an equational constraint is removed from the input set.

3.4 Bounds

A different approach to modify the projection uses *bounds*. Bounds are polynomial constraints of the form $b \cdot x + a \diamond 0$, with $a, b \in \mathbb{Z}$, $\diamond \in \{<, \leq, \geq, >\}$ and a variable x. They can be used to neglect some polynomials in the projection, namely those that are for all permitted values of their variables either always positive or always negative since these have no roots and are therefore not needed to determine the CAD. For polynomials that can be neglected due to bounds no successors (leading coefficient, discriminant, and resultants) need to be computed. This approach is described in more detail in [LSC+13].

4 Experimental Results

We implemented several of the modifications described in the previous section as part of the CAD module in our SMT solver SMT-RAT [CKJ+15]. These imple-

mentations are included in the publicly available version. To evaluate them we examine different combinations of the restricted and semi-restricted projection, as well as the simplification using bounds. We consider the following strategies: **B** uses only bounds to simplify the projection while **R** uses the restricted projection operator for as many as possible consecutive steps, starting at the first step. **BR** combines these two modifications. **BRI** extends it by allowing for an interruption of the restricted projection in the sense that it may be applied even when it is not applied in the step before. **BSI** employs the semi-restricted projection operator instead of the restricted one, otherwise, there is no difference to **BRI**.

For comparison we use **Default** that is the standard CAD based strategy in SMT-RAT. It uses a – supposedly less powerful – variant of the simplification based on bounds, but no optimizations using equational constraints. The modifications added to the regular CAD computations in the strategies **B**, **BSI**, and **Default** are provably sound, while for the other strategies no formal soundness proofs are provided yet. Additionally, all strategies are based on the projection operator due to Brown [Bro01] which is incomplete on certain inputs.

As benchmark problems we use the QF_NRA benchmark set from the SMT-LIB [BFT16] which consists of 12084 problems from 10 different applications. We used a time limit of 30 seconds per problem instance and allowed at most 4 GB of memory for every solver strategy and every input problem. The possible outcomes of a solving run are *sat*, *unsat*, *timeout* or *memout*. For *sat* and *unsat* the problem was solved correctly and is satisfiable respectively unsatisfiable. For *timeout* and *memout* the solver was unable to find a solution or determine unsatisfiability within the given time and memory limit.

We did not find incorrect results for any of the solvers on any input problem, despite the incompleteness of the projection operator used. For the projection operator due to Brown inputs that actually lead to incorrect results are known, but past experiments [VKÁ17] already showed that this benchmark set does not contain such examples.

Strategy	sat	unsat	timeout	memout
Default	4743	3939	3259	143
B	**4764**	3951	3225	144
R	4744	3962	3236	142
BR	4750	3964	3228	142
BRI	4752	**4039**	3151	142
BSI	4752	4038	3152	142

Table 1: Overall solver performances

The overall performance of each strategy is shown in Table 1. As expected all modified strategies solve more problems than **Default** and the improvements are mostly arising on unsatisfiable inputs. We assume that the reason is that the

solver can only take advantage of the modifications when a full CAD is computed since in that case fewer polynomials are needed. A full CAD is computed on an unsatisfiable problem to detect a theory conflict, which should occur more often for unsatisfiable problems. Satisfiable instances, on the other hand, are usually solved with only a fraction of the projection computed. We examine the number of solved problems for which theory conflicts occurred in Table 2. Most satisfiable problems can be solved without a single theory conflict. Also, most of the problems that could be solved additionally are problems where a theory conflict occurs.

Strategy	overall	thereof sat	thereof unsat
Default	3798	396	3402
B	3823	408	3415
R	3828	402	3426
BR	3835	407	3428
BRI	3912	409	3503
BSI	3911	409	3502

Table 2: Numbers of problems where a theory conflict occurred

The strategy **B** was the best on satisfiable problems, but also the one with the least improvement on unsatisfiable problems. The pruning of polynomials due to bounds reduces the size of the projection but essentially does not change the lifting phase as the removed polynomials provide no new samples anyway. The restricted projection operator further decreases the size of the projection but also removes samples from the lifting phase. As the restricted projection tends to remove polynomials of smaller degrees, this may actually be detrimental for the lifting phase and thus for satisfiable instances. We note that in practice there seems to be no significant impact of using the restricted projection on satisfiable instances.

The strategies **BRI** and **BSI** achieve the best results overall. Allow for interruptions in between the application of the (semi-)restricted projection operator improves the solver's performance on unsatisfiable problems significantly compared to the other variants. At least on this set of benchmarks, the differences between the restricted or semi-restricted projection are negligible.

Next, we take a closer look at the running times of the different strategies, based on the 8657 input problems that could be solved by all strategies. The results for the average running times are collected in Table 3. Compared to **Default** the simplifications by using bounds and the restricted projection operator do speed up the computations significantly. The best average running time has the strategy **BSI**, closely followed by **BRI**. This directly reflects the superior overall result of these two strategies.

Last we take a look at the memouts, as we can see that none of the modifications significantly changes the number of memouts. One would expect that

Strategy	running time in ms	for sat	for unsat
Default	705	108	1422
B	675	98	1368
R	683	113	1369
BR	673	101	1361
BRI	649	101	1308
BSI	649	101	1307

Table 3: Average running times

significantly reducing the size of the projections would mitigate memory issues. However, the modified strategies, in contrast to **Default**, never actually remove polynomials but merely deactivate them. This causes the solver to consume more memory which leads to more memouts. It is possible to delete these polynomials instead of deactivating them when using the modified strategies as well. However, we do not expect that to significantly improve the overall performance since this is only relevant for large problems that are often hard to solve anyway and are therefore likely to just result in a timeout instead. This gets even more likely due to the fact that deleted polynomials might have to be recomputed later.

We examined this by means of the **BRI** strategy and implemented the corresponding strategy with the deletion of polynomials, in the following referred to as **BRID**. In Table 4 the results for these two strategies are compared and they are indeed relatively similar. As expected more timeouts occur when using **BRID**, while in **BRI** one more memout occurs. Thus we observe that deleting polynomials even decreases the overall solver performance though the average running time on the problems solved by all strategies is nearly the same for both variants.

Strategy	sat	unsat	timeout	memout
BRI	4751	4039	3151	142
BRID	4736	4032	3175	141

Table 4: Comparison of deactivation and deletion

5 Conclusion

We examined the impact of restricted projection operators as proposed by McCallum in the CAD method on the performance of an SMT solver. After presenting several variants on how to use these in an actual implementation, we provided some experimental results for the implemented modifications. We can show that the performance improved especially for unsatisfiable inputs when the restricted or semi-restricted projection operators are used instead of the original

one. It, however, made no noticeable difference whether the restricted or the semi-restricted projection operator was used, though the repeated application is currently only validated for the semi-restricted operator.

We further investigated the difference of either deleting polynomials from the projection or only disabling them in the incremental setting during SMT solving. Though one could hope for a decreased memory consumption when deleting polynomials, this change did not make a significant difference.

Our ideas for future investigation concerning equational constraints mainly deal with making the best possible use of the restricted projection operator by applying this idea in as many levels as possible. One direction would be the use of the resultant rule [McC01] that allows propagating equational constraints. Another option would be the modification of the variable ordering heuristic depending on the equations that are present in the input. It would also be interesting to investigate whether a heuristic for the choice of which equational constraint to use for the restricted projection operator can be found. Currently we are using the equational constraint that is added first, however, the number of cells in the resulting CAD depends on the designated equational constraint as shown in [EBD15]. In that paper is furthermore shown how equational constraints can be used to make additional savings in the lifting phase, which could also be implemented in SMT-RAT.

References

[ADCTZ11] Pietro Abate, Roberto Di Cosmo, Ralf Treinen, and Stefano Zacchiroli, *MPM : A modular package manager*, 14th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE-2011) (Boulder, CO, United States) (Ivica Crnkovic, Judith A. Stafford, Antonia Bertolino, and Kendra M. L. Cooper, eds.), Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering, CBSE 2011, part of Comparch '11 Federated Events on Component-Based Software Engineering and Software Architecture, ACM, ACM, 2011, pp. 179–187.

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli, *The satisfiability modulo theories library (SMT-LIB)*, `www.SMT-LIB.org`, 2016.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, *Handbook of satisfiability*, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.

[Bro01] Christopher W. Brown, *Improved projection for cylindrical algebraic decomposition*, Journal of Symbolic Computation **32** (2001), no. 5, 447–465.

[CH91] George E. Collins and Hoon Hong, *Partial cylindrical algebraic decomposition for quantifier elimination*, Journal of Symbolic Computation **12** (1991), no. 30, 299 – 328.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám, *SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving*, Proceedings of SAT'15, LNCS, vol. 9340, Springer, 2015, pp. 360–368.

[Col75] George E. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, Automata Theory and Formal Languages, LNCS, vol. 33, Springer, 1975, pp. 134–183.

[EBD15] Matthew England, Russell Bradford, and James H. Davenport, *Improving the use of equational constraints in cylindrical algebraic decomposition*, Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation (New York, NY, USA), ISSAC '15, ACM, 2015, pp. 165–172.

[ED16] Matthew England and James H. Davenport, *The complexity of cylindrical algebraic decomposition with respect to polynomial degree*, Computer Algebra in Scientific Computing (Cham) (Vladimir P. Gerdt, Wolfram Koepf, Werner M. Seiler, and Evgenii V. Vorozhtsov, eds.), Springer International Publishing, 2016, pp. 172–192.

[GAB+17] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann, *Analyzing program termination and complexity automatically with AProVE*, Journal of Automated Reasoning **58** (2017), no. 1, 3–31.

[GGI+10] Sicun Gao, Malay Ganai, Franjo Ivančić, Aarti Gupta, Sriram Sankaranarayanan, and Edmund M. Clarke, *Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems*, Proceedings of FMCAD'10, IEEE, 2010, pp. 81–90.

[Hon90] Hoon Hong, *An improvement of the projection operator in cylindrical algebraic decomposition*, ISSAC '90 Proceedings of the International Symposium on Symbolic and Algebraic Computation (1990), 261–264.

[HR97] Stefan Herbort and Dietmar Ratz, *Improving the efficiency of a nonlinear-system-solver using a componentwise Newton method*, Tech. Report 2/1997, Inst. für Angewandte Mathematik, University of Karlsruhe, 1997.

[KS08] Daniel Kroening and Ofer Strichman, *Decision procedures: An algorithmic point of view*, Springer, 2008.

[Laz94] Daniel Lazard, *An improved projection for cylindrical algebraic decomposition*, Algebraic Geometry and its Applications: Collections of Papers from Shreeram S. Abhyankar's 60th Birthday Conference (Chandrajit L. Bajaj, ed.), Springer New York, New York, NY, 1994, pp. 467–476.

[LSC+13] Ulrich Loup, Karsten Scheibler, Florian Corzilius, Erika Ábrahám, and Bernd Becker, *A symbiosis of interval constraint propagation and cylindrical algebraic decomposition*, Proceedings of CADE-24, LNCS, vol. 7898, Springer, 2013, pp. 193–207.

[McC85] Scott McCallum, *An improved projection operation for cylindrical algebraic decomposition*, Tech. report, University of Wisconsin Madison, 1985.

[McC99] ———, *On projection in CAD-based quantifier elimination with equational constraint*, Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation, ACM, 1999, pp. 145–149.

[McC01] ———, *On propagation of equational constraints in CAD-based quantifier elimination*, Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation, ACM, 2001, pp. 223–231.

[MPP17] Scott McCallum, Adam Parusiski, and Laurentiu Paunescu, *Validity proof of Lazard's method for CAD construction*, Journal of Symbolic Computation (2017).

[VKÁ17] Tarik Viehmann, Gereon Kremer, and Erika Ábrahám, *Comparing different projection operators in the cylindrical algebraic decomposition for SMT solving*, Proceedings of the 2nd International Workshop on Satisfiability Checking and Symbolic Computation, CEUR Workshop Proceedings, vol. 1974, CEUR-WS.org, 2017.

[Vol15] Matthias Volk, *Using SAT solvers for industrial combinatorial problems*, Master's thesis, RWTH Aachen University, Germany, Aachen, 2015.

[Wei97] Volker Weispfenning, *Quantifier elimination for real algebra - the quadratic case and beyond*, Applicable Algebra in Engineering, Communication, and Computing **8** (1997), no. 2, 85–101.