Theory of Hybrid Systems Informatik 2
RWTH AACHEN UNIVERSITY

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

BACHELOR OF SCIENCE THESIS

PURGING SPURIOUS SAMPLES IN THE CYLINDRICAL ALGEBRAIC DECOMPOSITION

Daniel Alexander Heinen

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Eva Zerz

Additional Advisor:
Jasper Nalbach

Aachen, Date 12.11.2020

Abstract

The problem of solving constraint systems is a widely discussed topic in mathematics and computer science. Many ways to solve these systems have been developed. One way utilizing SMT solving is the use of the Cylindrical Algebraic Decomposition. As the complexity of the CAD is high, research to speed up CAD is relevant and still performed today. This thesis will explore the possibility to gain performance in the CAD by checking if samples are needed or could be skipped in the lifting phase. These checks will be done with multiple variants and different heuristics. They will also only be done on resultants and discriminants build in the projection phase. However, as later seen, the desired performance gain can not be achieved. This new alteration to CAD only worsens the performance of the used implemented CAD and reason will be given why. The main reason is that too few samples get deleted.

Contents

1 Introduction **1**
1.1 Problem 1
1.2 Related Work 1

2 Preliminaries **3**
2.1 SAT/SMT 3
2.2 CAD 4

3 Purging Samples **9**

4 Implementation **11**

5 Evaluation **13**
5.1 DeleteSamples 13
5.2 Evaluation 22
5.3 Summary 24
5.4 Full Lifting 25

6 Conclusion **29**
6.1 Future Work 29

Bibliography **31**

Chapter 1

Introduction

1.1 Problem

The problem of solving (non)-linear constraint systems with real numbers and variables is a mathematical problem with many applications. One application would be the solving of constraint systems for production machines. This thesis will take a closer look at constraint systems using polynomials compared to zero. Note that every (non)-linear constraint system can be converted into a constraint system of polynomials compared to zero using basic transformations.

Definition 1.1.1 (Polynomial)**.** *A polynomial in n Variables* $x_1,...,x_n \in \mathbb{R}$ *is:*

$$p(x_1,...,x_n) = \sum_{i=1}^{m} \prod_{j=1}^{n} x_{i,j} \cdot x_j$$

using $x_{i,j} \in \mathbb{R}$

Note that therefore polynomials can be either univariate ($n = 1$) or multivariate ($n > 1$). To solve these constraint systems (polynomials compared to zero) the *Cylndrical Algebraic Decomposition* (CAD) will be used[Col75]. This thesis then aims to analyze a possible improvement in the lifting phase, considering runtime, of the Cylindrical Algebraic Decomposition. Possibilities to delete not needed samples using properties of later introduced projection operators will be used to analyze performance gain or loss of that new variant of the CAD. Furthermore this thesis will only support partial checking of projection polynomials, namely resultants and discriminants and not the other projection polynomials.

1.2 Related Work

For related work refer to the works of the Theory of Hybrid Systems group at RWTH Aachen University, which researches and tries to improve CAD. In the following one examples will be given, because it contributed a big part to the CAD used in this paper. This contribution made by Gereon Kremer in cooperation with Prof. Dr. Erika Ábrahám was the implementation of a incremental CAD, which does not just naively execute the algorithm, but rather tries to save time by skipping steps if

possible[KA19]. Gereon Kremer contributed multiple works in an attempt to improve CAD.

Chapter 2

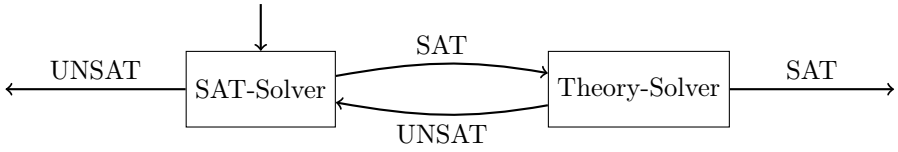
Preliminaries

2.1 SAT/SMT

Satisfiability Checking (SAT) describes the problem of checking the Boolean satisfiability problem.

Definition 2.1.1 (Boolean Formula)**.** *Let $b_1, ..., b_n \in \mathbb{B}$ be Boolean variables. Then a formula which uses $b_1, ..., b_n$ and $\neg, \wedge, \vee$ on those variables is considered a Boolean Formula.*

The Boolean satisfiability problem only consists of the checking of one Boolean formula for satisfiability. Following from that SAT in its most basic form can only be used for propositional logic. In most applications simple propositional logic is not sufficient to solve more complex problems. To solve more complex problems *Satisfiability Modulo Theories* (SMT) solvers were developed. SMT describes the solving of the same decision problem for solving satisfiability for a predefined theory. So this problem checks if a formula of theory expressions, which can be evaluated to true or false and gets connected using $\neg, \wedge, \vee$, where these operators are the commonly used operators for logic, can be satisfied.
These theories could be linear real arithmetic or nonlinear real arithmetic as an example. One possible way to solve SMT algorithmically is by using the DPLL(T), where T is a theory, framework [GHN+04]. The constraints become input in a DPLL(T) solver like seen in:

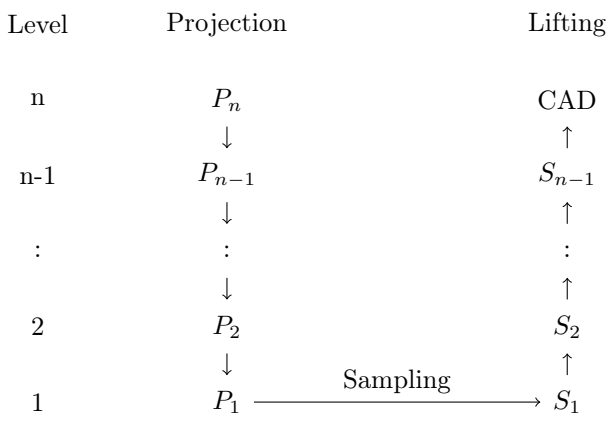


As seen a DPLL(T) solver joins together a SAT solver and a theory solver for the given theory. First the formula needs to be transformed in a way, so that there is a Boolean skeleton for the underlying formula using theory constraints. This Boolean skeleton then becomes the input of the SAT solver. The SAT solver tries to solve the skeleton and checks if a model could be achieved. If said model could be build, the information which constraints need to be true and which need to be false gets relayed

to the theory solver. The theory solver then checks if the theory constraints can be fulfilled in the way determined by the SAT solver. If the theory solver constructs a model, which satisfies the assignments, the SMT solver returns satisfiable since the original formula can be satisfied, because the boolean skeleton and the theory can be satisfied. On the other hand if the theory solver could not find a model it has to return a minimal set of constraints, which implied the unsatisfiability. With these the original set of constraints gets alternated to prohibit the SAT-Solver to find the same model, which was proven unsatisfiable, in the next run. If the SAT solver did not find a model the SMT solver returns unsatisfiable, because if the Boolean skeleton can not be satisfied no assignment using the theory can satisfy the formula.

2.2 CAD

The *Cylindrical Algebraic Decomposition* (CAD) is a method to solve a constraint system in (non-)linear real arithmetic. So in the context of SMT-Solving it can be seen as a theory solver. The specific input of the CAD are constraints of the form $p \circ 0$. Where p is an uni- or multivariate polynomial and $\circ \in \{<, \leq, =, \geq, >\}$. A constraint system then are multiple constraints, which all need to be satisfied. Note that the same variable can be used in multiple constraints. p gets restricted to have n variables. It follows that the assignments x of $p(x)$ are $x = (x_1,...,x_n) \in \mathbb{R}^n$. As defined, CAD only takes constraints compared to 0. Because of this, one intuition could be to only compare the regions where the polynomials do not change their sign. So the need for partial sets $R' \subseteq \mathbb{R}^n$, where p does not change its sign or is 0 arises. These regions are called *sign-invariant*. This works because if the polynomial p does not change its sign in a region it will either be $p > 0, p < 0$ or $p = 0$ for each point in the region on the given constraint compared to 0. For that reason we only want a way to generate the regions R' where all the polynomials, of the constraint system, do not change their sign or are 0. So it must hold for the region R', $\forall p(\forall r \in R'(p(r) > 0) \vee \forall r \in R'(p(r) = 0) \vee \forall r \in R'(p(r) < 0))$.
The regions can then be seen as cylinders, with cylinders as connected subsets $R' \subseteq \mathbb{R}^n$ in n-dimensions. These cylinders can be defined only using the roots of all polynomials, since these are regions themselves and the border points for sign-invariant regions. This is because the polynomial, which is zero at that point must be either positive or negative in the region itself. Then as seen before it is enough to get one point from this region to evaluate the answer of satisfiability for the whole region. CAD generates these cylinders using the following diagram:



As seen above CAD uses two phases Projection and Lifting[Col75].

2.2.1 Projection

The goal of the Projection phase is to reduce polynomials with n variables in the set P_n. The set P_n are the constraints of the constraint system. The reduction should aim at polynomials with $n-1$ variables in the set P_{n-1}. Note that in rare cases the reduction can skip a level to polynomials with fewer variables. So the goal is to eliminate one variable at each level of the Projection until only one variable is left. When using this reduction the important properties should still be observable in the polynomials in P_{n-1}. The important properties in the CAD are the roots of the polynomials and their relations to each other. As these are needed to construct the sign invariant regions in the lifting phase, since they are border/sample points for these regions. As later seen these sign-invariant regions are then used to solve the equation system. Now the roots of polynomials in P_{n-1} shall be the points in $n-1$ variables where these properties can be observed. The three properties/relations which needs to be saved are the following:

- First two polynomials have a common root.
- Second the function of roots of a polynomial
  can be observed in a local maxima/minima
  or the multiplicities of a single polynomial change.
- Lastly the function of roots of a polynomial behaves in way so that one parameter trends to +/- infinity while the other parameters undergo minimal change

All these properties assume that $n-1$ variables are fixed. In this thesis only the first two properties will be observed. The first property can be achieved by using the resultant of two polynomials.

Definition 2.2.1. *Resultant*
Let there be two polynomials p, p' of dimension n. The resultant r of these polynomials is the polynomial of lowest degree n' so that $x \in \mathbb{R}^n \wedge p(x) = 0 \wedge p'(x) = 0 \rightarrow r(x') = 0$, using $\forall_{i=1}^{i=n'}(x'_i = x_i)$.

This means that if the two polynomials which are used to construct the resultant, have a common root with fixed n-1 variables, this implies that the resultant will have

a root at the point of the fixed n-1 variables. A Resultant can be constructed by using Sylvester matrices[BL10].
To get the second property Discriminants are used.

Definition 2.2.2. *Discriminat*
Let there be a polynomial p of dimension n. The Discriminat of p is the resultant of p and the derivative of p using the variable corresponding to the level n.

From this follows that if the second property is fulfilled at x, then a root of the Discriminat can be observed at x' $\forall_{i=1}^{i=n'}(x'_i = x_i)$, if the Discriminant is of dimension n'.
How the third property is guaranteed will not be discussed here.
On one level these new polynomials will need to be constructed for every polynomial and every pair of polynomials. These resulting polynomials are then used as the polynomials of the next level (P_{n-1}). In turn the same operators are applied to these polynomials on level $n-1$. This gets done until every variable but one got eliminated. Following that P_1 is a set of univariate polynomials.
To get from P_1 to S_1 Sampling gets used. Sampling describes the process of finding the roots on every polynomial in P_1. This is necessary since all the important information for sign invariant regions are saved in those roots, as seen above. These roots get put together in a set $N \subseteq \mathbb{R}$. This set then gets supplemented by an arbitrary point in between every pair of neighboring roots and two points at the borders. So the points $s_{bl}, s_{bu} \in \mathbb{R}$ are selected as border points, so that $s_{bl} < n$, $s_{bu} > n$ for all $n \in N$. And for every n_1,$n_2 \in N$ using $n_1 < n_2$ and $\neg\exists n'(n' \in N \wedge n_1 < n' < n_2)$, a new point $s_{12} \in \mathbb{R}$ gets put in, so that $n_1 < s_{12} < n_2$. These are needed since the roots describe the border points, but the cylinders of the points in between are also needed to gather a valid solution. The resulting set is S_1.

2.2.2 Lifting

The Lifting phase now has to construct cylinders in $\mathbb{R}^n$ so that every sign invariant region is represented by a sample point of a cylinder. Furthermore the Lifting phase only gets the information saved in S_1 and the polynomials of the Projection phase. As seen in the diagram, the Lifting phase does this by going in opposite direction of the Projection phase. So instead of decreasing the number of dimensions/variables it increases them. This means that cylinders get build by adding one more dimension in each step. To get from dimension i to $i+1$, the need to construct S_{i+1} out of S_i arises. The lifting method used to lift the set S_i to S_{i+1} works like the Sampling method. It has to do that because the borders and important points in $i+1$ dimensions are encoded in the roots of the polynomials constructed in the projection phase for that level. It does the same with the exception that it uses the polynomials from P_{i+1} and puts each sample point $s \in S_i$ in each polynomial $p \in P_{i+1}$. Resulting from the fact, that since $s \in \mathbb{R}^i$ and every polynomial $p_{i+1} \in P_{i+1}$ and therefore has $i+1$ variables, the root finding is univariate after assignment of the i variables corresponding to the sample. Following that every root finding in the Lifting phase is univariate. The set of found roots for every s then gets supplemented like the roots in Sampling. The union of every new set from every $s' \in S_i$ is S_{i+1}. For S_{i+1} the same process gets executed. The resulting set in n dimensions $S_n = CAD$ is the set of sample points $s_p \in \mathbb{R}^n$ for every sign-invariant-region of the root constraint system. Based on the statements of the introductory part of CAD these points are enough to check every possible solution

to the constraint system. This means that only points from this set need to be checked to cover all possible solutions. So basically the problem of constructing n-dimensional cylinders got solved by only using univariate root finding and projection operators. Another advantage is that a model assignment to the variables (if it exists) can always be provided, as they are saved in the sample points of the CAD.[Col75]

Chapter 3

Purging Samples

As seen in the Lifting phase CAD builds a tree structure since every sample gets lifted with every polynomial of the level above. For that reason it could benefit the performance if there would be a way to delete samples, which can be determined as not needed to get a valid answer. However a precise performance gain of this method is difficult to predict. If one could make the assumption that the amount of roots, which get deleted is maximal, then the performance gain would be significant. Because the maximum number of samples which get deleted on level i originating from a sample on level $i-1$ is $|P_i| \cdot i$. This is because the sample on level $i-1$ gets lifted with every polynomial out of the set of P_i and each of these checks can produce at most i roots since every $p \in P_i$ is of dimension i and therefore can have a maximum of i real roots. Based on this formula, if a sample gets deleted on level $m-1 < n$, using n as the number of projection levels, then the number of deleted samples based on the first deleted sample would be $\prod_{j=m}^{j=n}(|P_j|) \cdot \frac{!n}{!(m-1)}$. This is just the multiplication of each level above the deleted level since for each sample deleted on a lower level the formula for the level above applies. This would mean an exponential growth of samples, but the underlying assumption can not be made generally. It can also be the case that the purged sample is the root for an empty tree, because there are no roots in any parent polynomials using the assignments of this sample. So 0 further samples would be eliminated. From these two points it follows that the number of further deleted samples $r \in \mathbb{N}$ of a sample on level $m-1$ is limited by $0 \leq r \leq \prod_{j=m}^{j=n}(|P_j|) \cdot \frac{!n}{!(m-1)}$. Note that these are only the direct samples based on the first sample. Samples which are filled in are not being considered for this estimation. So the results depend on the number of roots and where the purged samples are located in the lifting tree. But it should gain performance if enough samples are deleted, but no concrete assumptions of these numbers can be made. The question arises how samples which could be purged can be identified. To determine new samples the roots of the polynomial in the level above the current sample get used. These polynomials should only have roots at the important points for constructing sign invariant regions. As seen these points should be at the three properties mentioned in chapter 2.
However the operations, namely Resultant and Discriminant, only guarantee that there are roots at the given points. They do not guarantee that there are no additional roots which do not correspond to the given point in the parent polynomials. This follows from the fact that Resultants and Discriminants are only implying the properties, but there is no equivalent dependence. Furthermore the fact that poly-

nomials of degree n, in the real numbers, always have n roots in the complex space, is an important factor why these properties can only be seen as implications and not equivalences. The property above would be fulfilled if the only real roots of the polynomial are the roots at the given points. In that case the rest of the roots would be complex. However if at least one of the complex roots, which are not desired, is in the real space, it will be seen as an important point and will be lifted to construct a cylinder, which would not be necessary. So as discussed earlier it could benefit the runtime if the unnecessary samples would be filtered. This can be done by utilizing the properties defined in the definitions of the Resultant and Discriminant. So one way to filter unnecessary roots would be to check if the parent polynomials of the polynomial which are used for lifting have a common root at that point in the case of the Resultant, since only these should be roots as important points. When it was determined that the parent polynomial was a Determinant the same gets done for the polynomial and its derivative. This can be done by getting the sets of roots of the polynomial and checking if they have a common member. If they would not have a common root no important property would be at this point and one could purge this sample and therefore save the whole part-tree, with this sample as a root. Deleting in this case implies not lifting the sample, so to not use it further. Also if the sample was not deleted the determined roots can be later used as seen in the implementation.

Chapter 4

Implementation

For the implementation of CAD, SMT-RAT developed by the THS group at RWTH Aachen University was used as a foundation[KA18]. SMT-RAT is a library which provides multiple solvers and strategies in SMT-Solving. One of these solvers is a CAD solver. Another example would be a Max-SMT solver. Furthermore it can provide means for quantifier elimination. In SMT-RAT every solver can be executed using multiple strategies. These strategies alternate the way in which the solver solves the input file. For example in the CAD solver Projection and Lifting priorities can be manipulated in the strategies. As a strategy, if not declared otherwise, CADOnly was used. For further explanation on this strategy refer to the SMT-RAT Documentation. As SMT-RAT was used an incremental CAD was used. The incremental CAD changes the lifting and projection so that performance gains can possibly be achieved. In the lifting the incremental CAD stops after finding one satisfying sample, when given a satisfiability problem like given here. This means that not the whole CAD is constructed but rather a part until a satisfying sample is obtained. Furthermore it introduces a priority queue for lifting samples, which enables the user and strategy to optimize the order in which the samples are lifted. In the Projection a similar approach is implemented. A Projection queue gets used so that one entry polynomial can be projected at a time. After that the Lifting gets checked for a possible solution. If a solution would be found the remaining projection would not have to be computed and therefore time would be saved compared to the standard CAD where the whole Projection needs to be computed. In addition heuristics, which estimate the computational cost of one projection polynomial can be applied. So in summary the incremental CAD can gain performance because computations can be saved by finding a solution to the problem early.[KA19] Furthermore for the implementation of mathematic algorithms, for example root finding and representation of numbers/intervals, the CArL Library was used[RIT20]. Alternations were made to SMT-RAT to enable the purging of samples. Note that no alternations were made to algorithms used in the CArL library. For that reason they are not in the consideration for the results by the modified SMT-RAT library.
In the standard implementation of the purging of samples the method LiftSample of SMT-RAT got alternated. First a utility was added to pass down the parents of the polynomial, with which the sample should be lifted. These information included the parents itself and furthermore information if the polynomial was created by building the Resultant or Discriminant of polynomials or if another method was used. In the

case that the polynomial is a Resultant or Discriminant the parent polynomials are coded as well, so that they can be found in the Projection. This is needed to perform the necessary checks later. Using this information the algorithm got coded by first deciding if a sample is eligible to be purged. A sample is eligible, if it is constructed out of the Resultant or Discriminant of parent polynomials. If it is eligible and in the case of the sample being a Resultant, the roots of the parent polynomials of the lifting polynomial are determined. They can be found using the sample level and the coded ids of the parents. Following that the two sets get compared element by element. So in this implementation this part would have a runtime of $r1 + r2 + m * n$ with $r1, r2$ being the time needed to determine the roots of the polynomials. m,n are the number of elements in the root sets. Note that also minor overhead gets added. For example the checks or additional for/while loops, which run their loop checks every pass. The same goes for the case that the lifting polynomial is a Discriminant. But now only one parent polynomial is given and a derivative of this polynomial needs to be determined. Again the roots get determined and compared. $r1, r2$ still are needed, but can get expensive and are the most costly part of the extension proposed in this thesis, so the reduction of time for root finding and the reuse of the root sets is important. For that reason storage is traded in for runtime. This is possible, because if the sample is not purged in most cases the parent polynomials are in the level above the new sample and because of that roots for the parent polynomials using this sample, as the assignment for the fixed variables, are still needed. These are exactly the roots used in the process of determining if the sample is purged. Note that this is applicable to both polynomials in the Resultant case but in the Discriminant case the roots of the derivative can not be used and could therefore be a performance problem. The roots that can be saved are stored in an array. Specific roots for a samples are stored using the sample level, the id of the corresponding parent polynomial and a newly introduced id for each sample. This id simply counts the samples in a level and assigns the ids incremental to the lifting of the samples.
In the following the basic algorithm and alternations will be analyzed based on benchmarks results. The benchmarks used for every result, which are to follow, are all part of the SMT-LIB. Concretely QF-NRA out of SMT-LIB was used[SLB20]. In every run, every smt2 file out of QF-NRA was executed. They were executed using a timeout of 3 Minutes, a maximum storage of 1 GB and using the benchmax executable available in SMT-RAT[KA18]. The Benchmarks were executed on 4 processors 2.1 GHz AMD Opteron (12 cores) and 192 GB RAM.

Chapter 5

Evaluation

5.1 DeleteSamples

DeleteSamples will be the name for the modified Algorithm, which was described earlier, in the following figures and analysis.

Figure 5.1: File Comparison

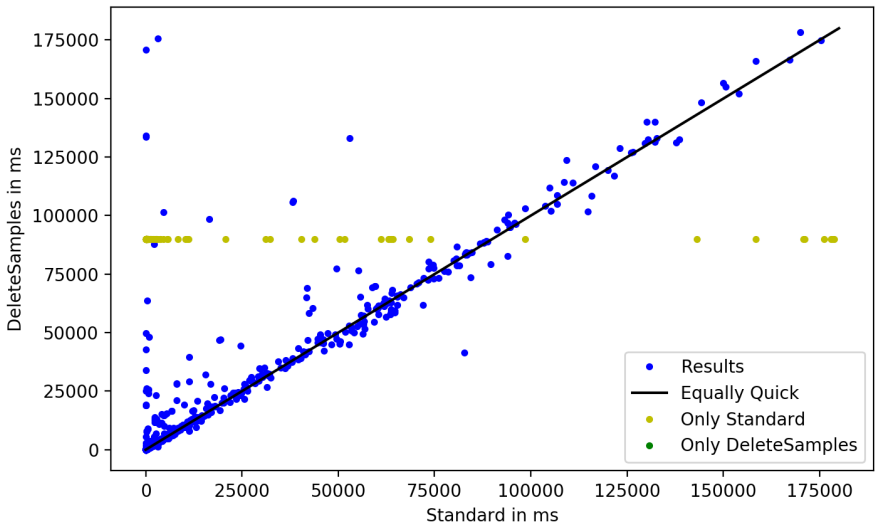
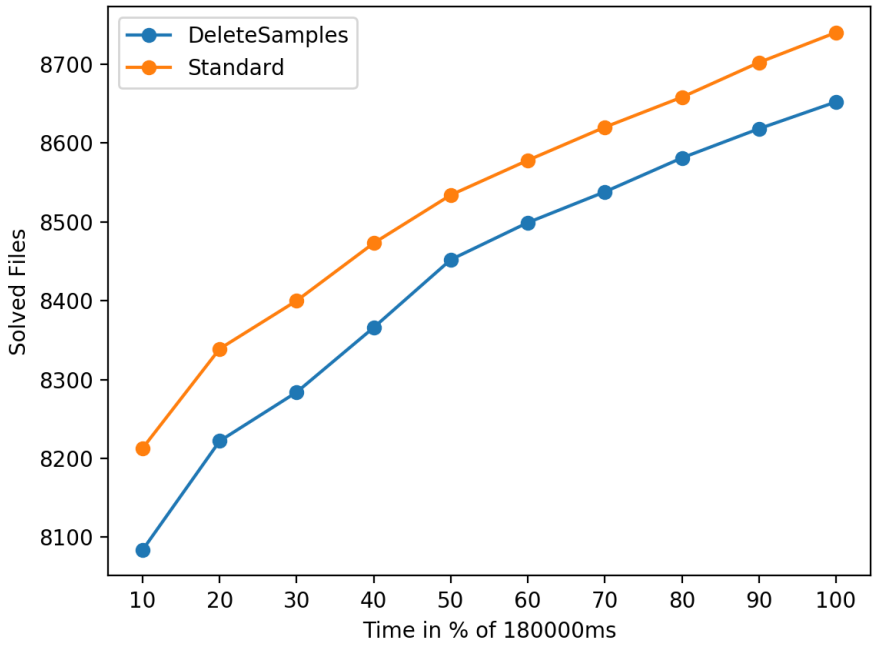


Figure 5.2: Performance Graph



Figures

First a short explanation what these Figures (1,2) depict and how they are to be interpreted is given. They will be used for every following comparison and analysis in this thesis.
Figure 1 describes the relation of runtime of two versions of SMT-RAT. In this case this would be DeleteSamples and the Standard algorithm implemented in SMT-RAT. The 'Results' points in this graph describe the files which finished with a valid result for both algorithms. This means that both variations resulted in either SAT or UNSAT. Then the resulting point for that file will be at the point (x,y) with $x = t_1$ and $y = t_2$, with t_1 = runtime in the standard variation, t_2 = runtime in DeleteSamples variation. This means that if a point is over the Equally Quick line then Standard was faster than DeleteSamples for that file. The same goes for under the line. Equally Quick just describes a line so that x = y for every point (x,y) on the line. This then would mean that the files were equally quick.
The yellow and green dots describe the fact, that only one variation managed to get a result in 3 Minutes. They are located at the halfway mark for the variation that did not finish and at the finish time of the variation that actually managed to finish them. In the specific example above this means, that DeleteSamples did not manage to solve files, which the Standard algorithm did solve.
Figure 2 is a Performance Graph. This means that these figures depict the solved files in a percentage of time of the maximum runtime (3 min = 180000ms). So for example the point at 20 % describes the number of all files solved in 20% of the time

Only DS		Only Standard	
Time	Mem	Time	Mem
0	0	121	4

Table 5.1: Only solved: Time/Mem

of the timeout.

Results

So from a mathematical standpoint DeleteSamples should bring performance if enough samples get deleted to make up for the loss which gets created during the derivation and the consecutive root finding of the polynomials which got projected using the Discriminant. This is because every other determining of roots needs to be done by the Standard algorithm too, since the sample which gets checked will need to be lifted using the original polynomial, in the case of a Discriminant, or both polynomials, in the case of a Resultant. Therefore there should be no time loss, when comparing the Standard implementation and DeleteSamples. Note that further overhead got introduced in the implementation of the algorithm, but this overhead is negligible since it is mainly assignments or if/else checks. Especially on larger files it should not influence the runtime. The table pictured in Table 1 however, gives an overview over how many files were gained/lost between the two algorithms. This means that if a file finished in one variant but not in the other it would be added to that column. Furthermore it is specified if the other variant time out or if it failed because of a an excess in memory.
So on the first observation of Figures 1,2 it gets clear that DeleteSamples does not gain any performance improvements. In fact it worsens it. On the other hand Figure 2 shows that some performance gets lost in the lower times, but its comparable when approaching times larger than 1 second. Comparable here means that the curves are similar. So the difference of solved files stays almost the same. But when looking at the numbers from Table 1, one can see, that 125 files which get solved during the standard algorithm do not get solved using DeleteSamples. Even though this indicates that the performance is worse. this also indicates that the swap of runtime for storage can be made, because the memouts do not rise for a huge amount. Note that this assumption can at least be made if 1 GB of Storage is available to solve every file. The results may differ if less storage is available.

File Analysis

In the following analysis few representing files will be more thoroughly analyzed to determine why the implementation does not indicate the performance gain that the mathematical problem would implicate.
First files which are solved within 5-10s will be observed. Specifically there will be the restriction, that the time of Standard can not deviate more than 5% of the time of DeleteSamples. In theory there are multiple possibilities how the two methods produce a similar output. As a first possibility, there could be no complex root finding and derivation, so root finding problems and the derivation do not take a large amount of time or time at all. This would imply that the time loss of performance

implied above would be very small and the algorithms produce a similar output. The other possibility is that there is quite an amount of overhead and time lost in the derivation and root finding. In that case, to achieve similar results, a few spurious samples need to be found to cancel out the overhead.
On all observed files it was the case that the overhead created by DeleteSamples was not enough to alter the result of the file. An important remark here is that only a selective part was observed more thoroughly and there could be the possibility that other files still delete samples to cancel out the overhead. But if that is the case they are the strong minority in the observed state.
Another interesting case which will be observed is that one variant finishes fast and another takes much more time or does not finish at all. In the case, that DeleteSamples was much faster than Standard it should be the case that enough samples were deleted to cancel the overhead and to gain performance. However as seen in Figure 1 and in Table 1 there are not many files where this is the case. Concretely there are two files
("20180501-Economics-Mulligan/MulliganEconomicsModel0061c.smt2"/
"zankl/matrix-1-all-21.smt2") which were almost 50% faster in DeleteSamples than in Standard. Note that only files which ran longer than 1s were considered for this selection, since a 50% gain in runtime is not much and easily achievable under 1s. On further analysis the zankl file performed better in the benchmarks than in separate analysis where only the file was checked. Because the zankl file was almost equal in separate analysis and took longer than in the benchmarks. Nonetheless in zankl 1290 samples were deleted, but this resulted only in a gain of 2038 not lifted samples in the new variant of 18354 in the old variant. This shows that many samples must be deleted in the upper levels since a deletion in the lower levels would mean that more samples would not be lifted. The Mulligan file performed similarly to the results in the benchmarks. This also shows in the case of the not lifted samples in comparison to the old variant. The new variant does not lift 7671 samples which the old one does lift. The standard variant lifts 19980 samples. A surprising fact is that DeleteSamples achieves these 7671 not lifted samples with only 86 samples which were deemed not needed. This means that these 86 samples have to be at a lower level since then the whole tree structure above those samples does not need to be build and many not lifted samples can be achieved. So it follows that for a gain in performance it is not sufficient that a considerable number of samples get deleted, but that they also should be deleted as low as possible. These results also show that not enough samples get deleted to gain performance in the other files.
The inverted case, that Standard was much faster than DeleteSamples, should only be possible if no samples get deleted and the overhead, which gets introduced considerably worsens the performance. So the expectation for these cases would be that there are derivations and root findings for these derivatives, which are very time costly. On further observation it gets clear that one problem that arises with the CAD in its implemented variant is that some roots get determined, in the checking process, which do not get used later in the run because the Standard algorithm arrives at a result before these result would have been needed. So for example if a file ends in SAT before further samples would be lifted, but in those samples which still were not lifted there was a costly root finding task, then this task would only be executed in the DeleteSamples algorithm, if it was used to check the necessity of a sample. This is caused by the incremental CAD. In theory this problem should only occur with satisfying files. Therefore there will be a comparison on runtime based on the result

SAT		UNSAT	
Standard	DS	Standard	DS
2297	1555	938	221

Table 5.2: Core Times

DeleteSamples		Standard
Samples Not Needed	Samples DS	Samples Standard
13951984	725284242	777976935
Not Needed/DS	DS/Standard	
1.92%	93.72%	

Table 5.3: Sample Numbers

of file in the following.

SAT-UNSAT

In contrary to the statistics talked about above, where the runtime of the whole file was the statistic observed, in this following comparison only the times of the CAD Core itself will be considered. As a quick implementation note, these times were made using the system time and are therefore rounded to seconds, so very precise results as seen above are not possible. Also a fresh run of the benchmarks was used. This run used the same parameters. Compared were files that both finished in SAT or both finished in UNSAT as well as files which only finished in one variant.
The concrete numbers for this comparison can be found in Table 2. In this Table (2) the number indicates how many times an algorithm was faster under a certain result. So in contrary to the expectation, which was that in the case of an UNSAT file the runtime would be close to equal in the core itself, the core time in the files, which are UNSAT are much more onesided then in the SAT case. This indicates that not enough samples get deleted, because even though the UNSAT files need to build all samples not enough samples get deleted to compensate for the overhead. On the other hand this result can also indicate that, if enough samples get deleted in the SAT case and 'trees' which will be deleted, would still be build in the Standard variant.
As the previous results indicate, there are probably not that many samples which get deleted. For that reason a analysis of the deleted samples and on which level they occur will follow. These results will be based on again separate benchmarks, which were done in the same way as the benchmarks above.

Sample Analysis

The numbers in Table 3 represent the numbers of the respective samples in the benchmarks and how many samples were not needed in DeleteSamples. Note that if a sample was not needed in DeleteSamples, it is still counted in the sample count of DeleteSamples. An important detail here is, that not all these samples which were not constructed can be tracked back to deleted samples. So probably not all of the 6.28% gain of DeleteSamples comes from deleted samples. This gets further reinforced by the fact that the number of deleted samples of all checked samples is under 2%. As

DeleteSamples		Standard
Samples Not Needed	Samples DS	Samples Standard
430165	14905635	15094444
Not Needed/DS	DS/Standard	
2.89%	98.75%	

Table 5.4: Sample Numbers (Both Finished)

Resultants		Discriminants	
Not Needed	Sum	NotNeeded	Sum
13507947	174248519	444307	1193144
7.75%		37.22%	

Table 5.5: Sample Numbers (Resultant/Discriminant)

seen earlier Standard finishes files in which DeleteSamples does not finish because it needs to do an expensive root finding task. In this case Standard has more samples even though no samples were deleted. For that reason a comparison of using only files were both finished is needed.

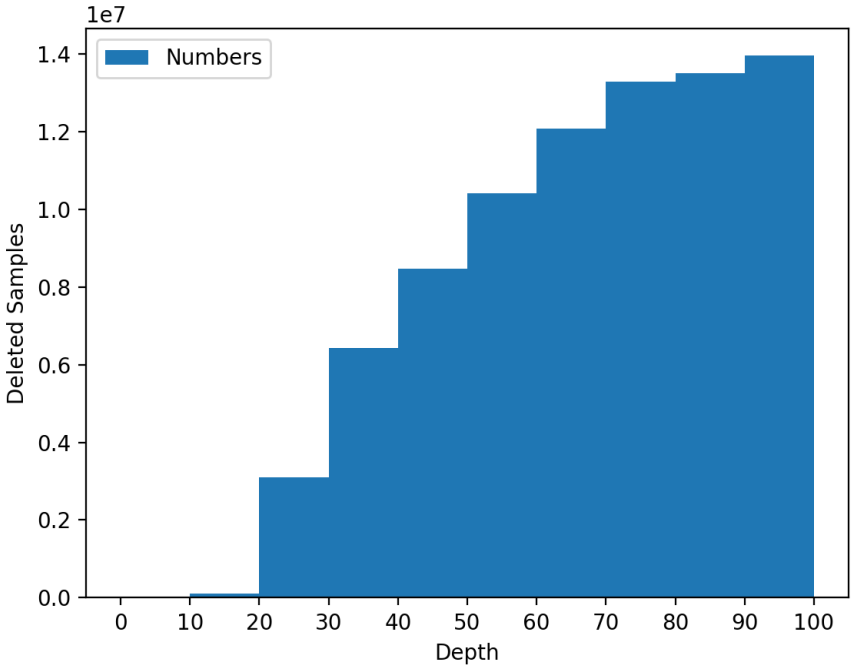
As seen in Table 4 the percentage of samples which get deleted is almost the same, but the number of the relative amount of all samples grew a significant amount. For this reason it is possible to say that most gains, in terms of not lifted samples, which are shown in table 3 come from files which needed to do an expensive root finding task rather than from sample deletion.

In the following figures of Resultants and Discriminant numbers will be observed separately.

Table 5 shows the numbers of samples lifted with Resultants and Discriminants respectively and how many were deleted. For this table all files were observed. It shows that Discriminants seem to have a higher chance at purging samples but are used far less to lift samples than Resultants. For this a reason a heuristic will be implemented which uses only Resultants (refer to 5.1.1).

As seen before the level is also a deciding factor in the effectiveness of this new algorithm. The average level of a deleted sample is at 54% of height of the Lifting tree. Note that for this number only files which actually deleted samples were considered.

Figure 5.3



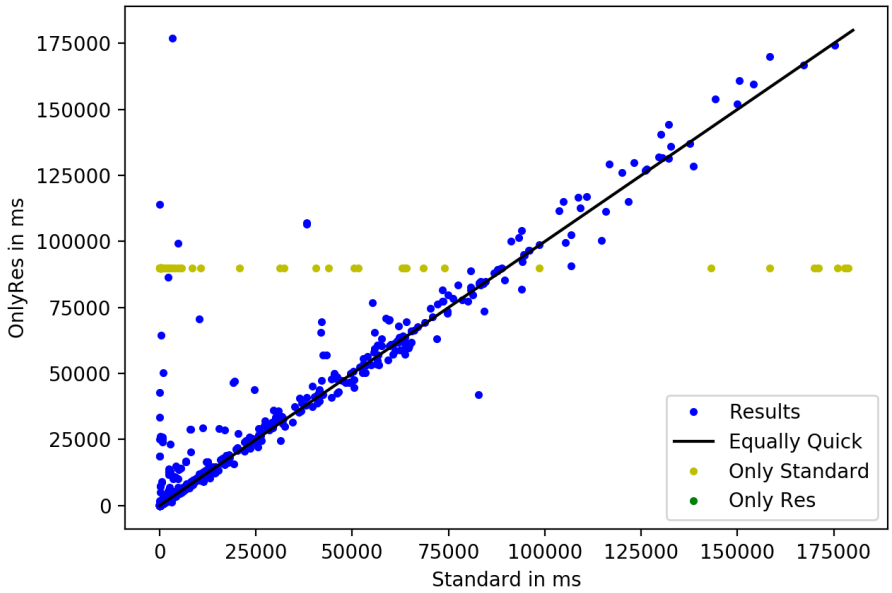
The Figure above shows a more detailed description of how many samples got deleted up to a certain level. The Depth is given as %. The bin at 100 is equal to the number of all deleted samples. This shows that almost no samples get deleted at the bottom levels, which would be desirable to gain more performance. Level in this context is always meant as the level as seen in the Lifting of the CAD in Chapter 2. So it is corresponding to the number of variables in the polynomial which was used to lift the sample. Also most of the purged samples are located in the medium levels. Furthermore there are not many samples which get deleted on the upper levels, so this implies, that whole part trees are more often purged than just one sample. When talking about low/medium/high levels it is important to notice that for one file a high level could be level 4 and for another file it could be level 50. So this graph just gives an intuition of the location of deleted samples. Although this could mean that the design of Resultants and Discriminants works better on polynomials with less variables, than with more variables. This assumption could be made because the levels will balance out on average over the 12000 files. The analysis on this topic will not be discussed in this thesis.
Based on these results and the results before a heuristic will be employed, which could improve the performance of the variant. This heuristic will be the usage of only Resultants for checks and ignoring Discriminants. At first this seems counter intuitive, because the percentage of not needed samples is larger in the Discriminant checks. However since the Discriminants are a lot less frequent than the Resultants the percentage is higher, but the absolute number of deleted samples is much lower. Furthermore the Discriminant check introduces much more overhead than the Resul-

tant check. For those reasons it may not be feasible to check Discriminants in the picture of all files. Moreover this new heuristic should improve performance since more results of the Resultants check can be reused and the performance of 7.75% deleted samples suggest that it will still be worth.

5.1.1 Only Resultants

As a result of the previous analysis a further variant can be implemented and will be analyzed in the following. The Only Resultants variant only checks Resultants if they are needed. This means that for every Discriminant no check is performed and they are viewed as needed. By doing this the overhead of the check for the Discriminant is eliminated. This leaves only the Resultant overhead which in theory should not influence the performance as much.

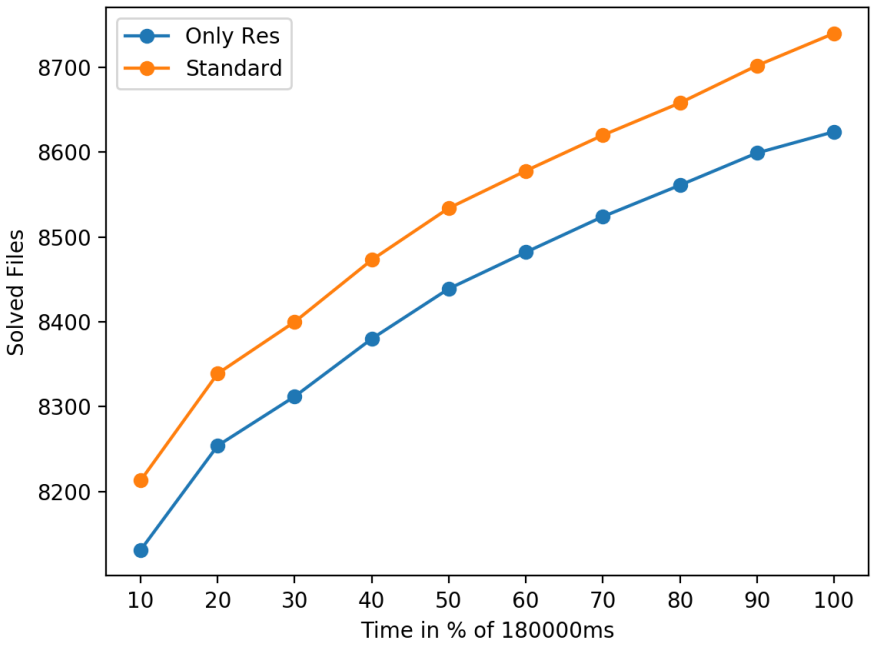
Figure 5.4: File Comparison



Only ResOnly		Only Standard	
Time	Mem	Time	Mem
0	0	104	5

Table 5.6: Only solved: Time/Mem

Figure 5.5: Performance Graph

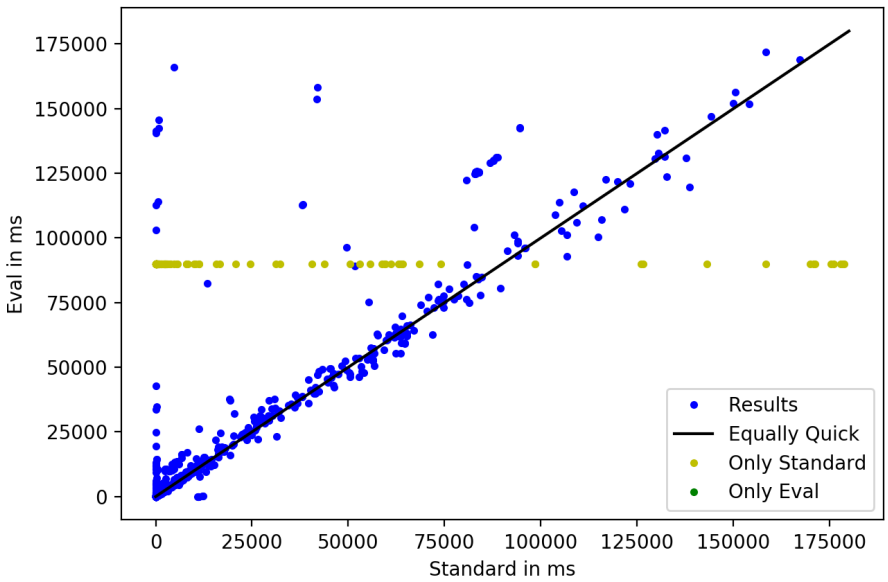


For explanation on the type of Figures (4,5) and the Table (6) refer to DeleteSamples. From these Figures it gets clear that these variant gains performance over DeleteSamples but still loses in comparison to the Standard variant. This can be explained for the reasons given in DeleteSamples. The performance gain is explained above and can be observed in the files. So it is the case that so few samples get deleted that the saving of the overhead introduced in the Discriminant check improves performance. One important fact to note is then that checks for Discriminants seem not feasible in the views of this result.

5.2 Evaluation

Another big change that was made, was in the evaluation of roots since this is a big part of the time loss and problem in the CAD. Since roots for Resultants and Discriminants and their parents respectively will need to be calculated, a time gain here would improve the overall performance of the entire procedure. In the original variant (DeleteSamples) the roots for the parent polynomials were calculated and then compared element by element. For this variant a partial evaluation of polynomials of the CArL library was used. The partial evaluation works by creating a construct which saves the assignments saved in the new sample which is to be lifted since every sample encodes assignments for each variable as designed by CAD. Then the roots of one parent polynomial get determined and get put in the same structure which then evaluates the polynomial based on the assignments. After assigning these variables there can be no more unassigned variables in the parent polynomial. This then enables to save the root finding for one parent polynomial. But it prevents the saving for both parent polynomials.

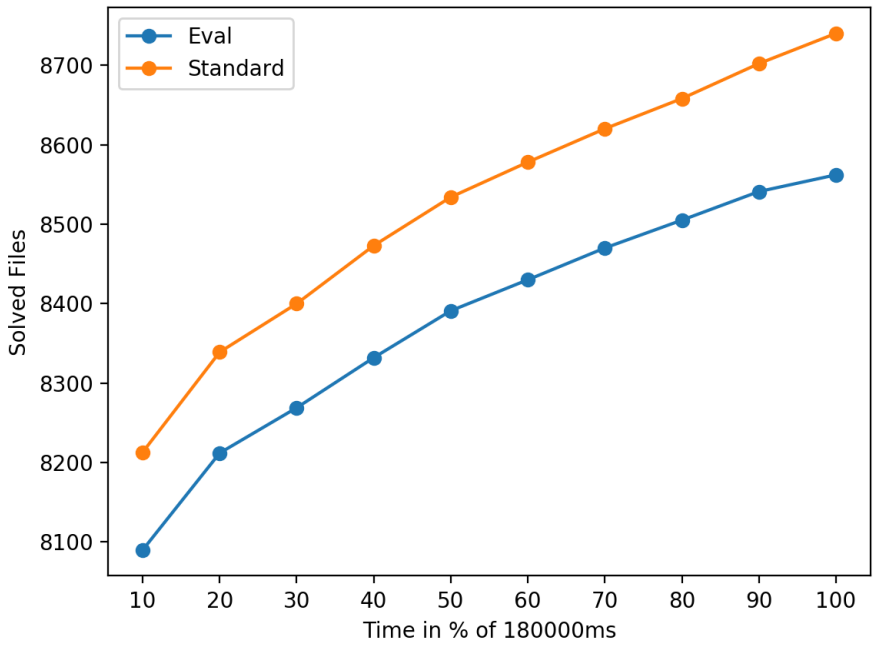
Figure 5.6: File Comparison



Only Eval		Only Standard	
Time	Mem	Time	Mem
0	0	156	3

Table 5.7: Only solved: Time/Mem

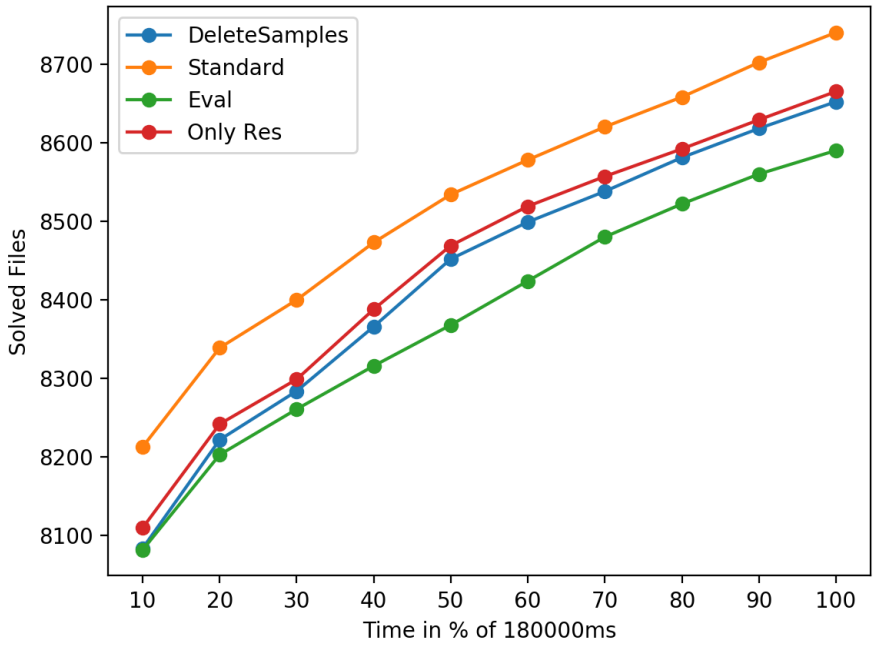
Figure 5.7: Performance Graph



The Figures (6,7) and Table (7) depict the same comparisons as in DeleteSamples. For further explanation on what the Figures and Table depict refer to DeleteSamples. As seen in the tables the evaluation variant performs even worse than the DeleteSamples variant. This can be seen especially in Figure 7 and Table 7. First as seen in Table 5 the evaluation variant loses even more files compared to the standard variant and does not gain any files. Furthermore it shows worse performance in Figure 7. All this also indicates that not enough samples get deleted for this variant to be feasible. Because the only way that Evaluation shows a such a worse performance than DeleteSamples is when almost no samples get deleted and therefore there is no time saved in the evaluation, since the roots will be determined anyway. If samples would get deleted determining of roots and therefore time could be saved as the evaluation should be faster than root finding. This is then consistent with the analysis of the samples in DeleteSamples. The analysis in DeleteSamples can also be applied here since this change in root comparing method does not influence the behavior of deleted samples, because if a sample gets deleted in DeleteSamples it will be deleted in Evaluation.

5.3 Summary

Figure 5.8: Performance Graph



This Figure shows a comparison of all variants which got introduced so far. All these variants work on the incremental CAD. So in summary all variants loose performance in comparison to the standard variant implemented in SMT-RAT. The Only Resultant variant gains performance on DeleteSamples but it is only marginal. This is because too few samples get deleted because of the reasons stated above. Furthermore the evaluation off roots does not gain performance as Eval is worse than any other implemented variant.

5.4 Full Lifting

Full Lifting describes the same algorithm as in DeleteSamples, but this time constructs all cylinders in the CAD. This means that even if an result is obtained every other cylinder will still be constructed. So basically a basic CAD in contrast to the incremental CAD was used. In return the problem that the implemented CAD stops after finding an answer in DeleteSamples will be canceled out. Because now the CAD will continue and therefore every root resulting from a Resultant check can be reused.

Figure 5.9: File Comparison

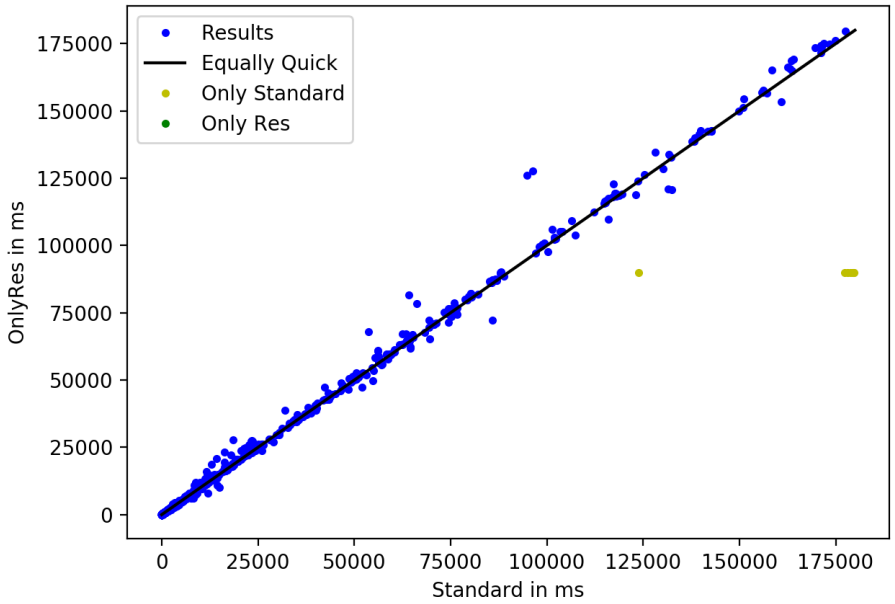
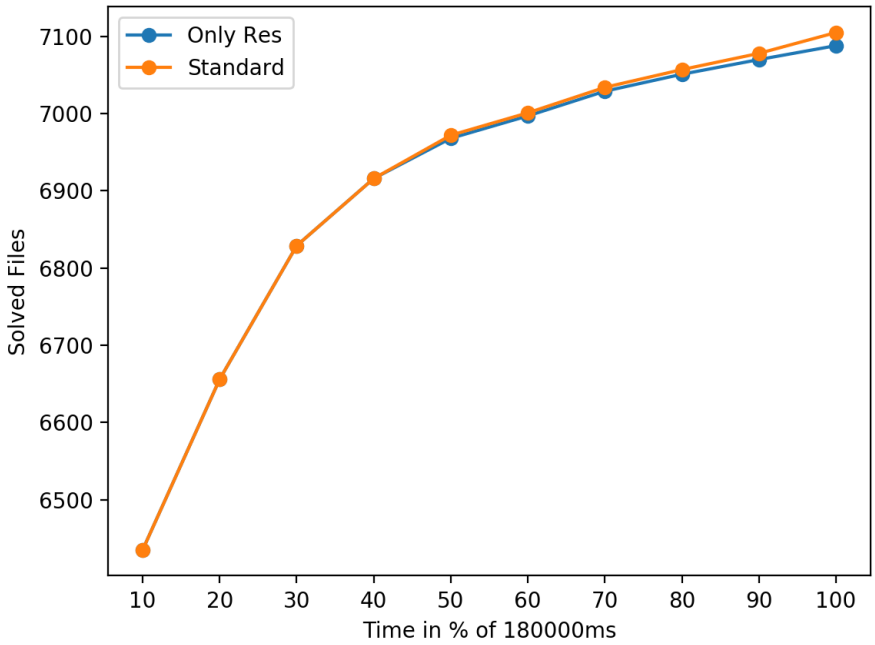
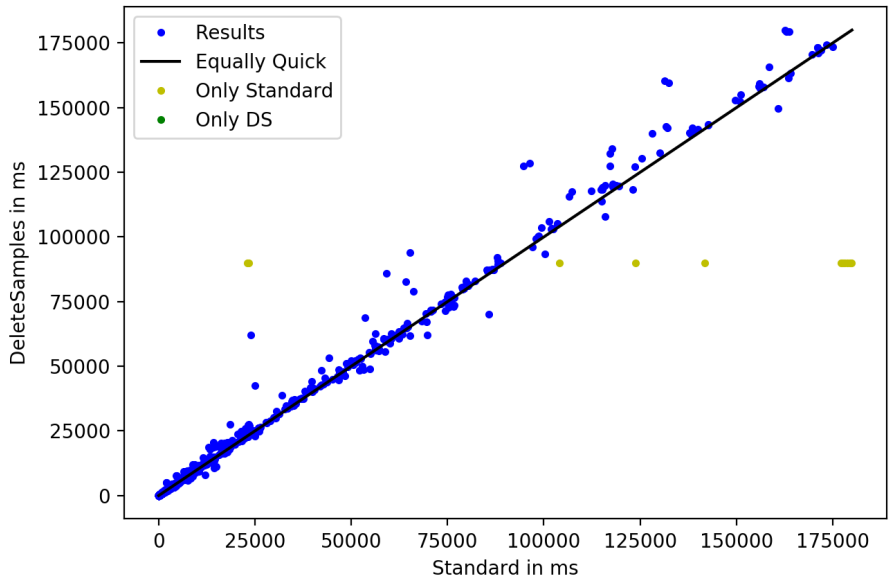


Figure 5.10: Performance Graph



Depicted in Figure 9 and 10 are again the same graphs as shown before in this thesis. The immediate impression is that the performance gain/loss got smaller. Furthermore there is pretty much no difference in the two variants. This reinforces the fact that not enough samples can get deleted from checks only using Resultants or checks at all. Because not enough samples get deleted to balance out the minor overhead which is created by the checks there is still some performance loss. Minor overhead refers to the added if/else checks or assignments which were added. Any major overhead, root checks for example, can be reused as explained before, so no time loss/gain is produced in these. So therefore the minor overhead is the only overhead in this variant. In contrary if enough samples would be deleted there should be a performance gain because no additional overhead is created.

Figure 5.11: File Comparison



For comparison Figure 11 shows the time graph for the standard algorithm in comparison to DeleteSamples in the Full Lifting variant. Now the benefit of reusing everything can no longer be relied upon. Since Discriminants are checked now, additional overhead is created. This immediately shows in the performance which is now worse as expected. Again this points to few sample deletions, as the increase in overhead is immediately observable and no gains by further sample deletions are made. However it is still better than the version of DeleteSamples using the incremental CAD. This again shows that the incremental CAD is counter productive to this DeleteSamples algorithm, because not every major overhead which gets introduced can get reused (at least in the Only Resultant variant).

Chapter 6

Conclusion

This thesis aimed at exploring multiple variants of the Cylindrical Algebraic Decomposition. These variants purged samples, which were created in the lifting of polynomials in the lifting process of the CAD, but which were not needed because they were not at an important point. To explore this possibility the SMT-RAT library was extended to allow for the necessary checks. The checks were performed on Discriminants and Resultants. However the results showed that this new variant and implementation had two problems. First too few samples could be purged to cancel out the overhead and second the construction of the implemented CAD blocked assumptions which could be made based on the mathematical algorithm. Only 2% of the samples were found not needed, which proved to be not enough to cancel out the overhead. Since these first results were not improving the performance multiple alternations of the basic algorithm were implemented.
The first alternation was the checking of samples stemming from Resultants, but not from Discriminants. This alternation gained performance compared to the new variant, but was still considerably slower than the standard variation which does not check samples. The second alternation was the implementation of a different method to find roots, the evaluation method. This proved to be slower than the DeleteSamples method, because it created more overhead which could not be compensated because to few samples were deleted. The third alternation was an alternation to the whole CAD so that every sample needs to be lifted. The result of these benchmarks were that again not enough samples were purged. It also showed, that when observed alone, the Resultant variant proves faster than the standard variant even in the Full Lifting.
In summary one can say that purging samples, at least in this version, is not enhancing the performance of the CAD.

6.1 Future Work

One possibility to further test this variation would be to implement a check for every possible polynomial which gets created during the projection. In addition this thesis only took a look at two forms of polynomials used in the projection. It could be extended by performing checks for the other projection polynomials too. Furthermore every enhancement of the CAD would also enhance this new variant. So for example an improved root finding method would also benefit this variant. Additionally if a

higher percentage of deleted samples could be achieved, for example through different heuristics, a new try at implementing this variant would be considerable. Because as seen this new approach does not gain performance for the fact that to few samples are deleted.

Bibliography

[BL10] Yuval Bistritz and Alexander Lifshitz. Bounds for resultants of univariate and bivariate polynomials. *Linear Algebra and Its Applications*, 432:1995–2005, 04 2010.

[Col75] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975*, pages 134–183. Springer, 1975.

[GHN+04] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast Decision Procedures. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification*, pages 175–188, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[KA18] Gereon Kremer and Erika Abraham. Modular strategic SMT solving with SMT-RAT. *Acta Universitatis Sapientiae / Informatica*, 10(1):5–25, 2018.

[KA19] Gereon Kremer and Erika Abraham. Fully incremental cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 100, 07 2019.

[RIT20] RWTH-I2-THS. CArL. `https://smtrat.github.io/carl/`, 2020. Accessed 8-10-20.

[SLB20] SMT-LIB-Benchmarks. QF-NRA. `https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_NRA`, 2020. Accessed 8-10-20.

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Heinen, Daniel Alexander | 377182

Name, Vorname/Last Name, First Name | Matrikelnummer (freiwillige Angabe) Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Bachelorarbeit mit dem Titel
I hereby declare in lieu of an oath that I have completed the present Bachelor thesis entitled

Löschen der falschen Beispiele in der zylindrischen, algebraischen Dekomposition

Purging Spurious Samples in the Cylindrical Algebraic Decomposition

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting)
erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date | Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date | Unterschrift/Signature