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

AUTOMATED EXERCISE GENERATION FOR SATISFIABILITY MODULO REAL ALGEBRA

Antoniu-Paul Filip

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr.-Ing. Ulrik Schroeder

Aachen, October 14, 2021

Abstract

Over the years the selection of procedures present in SMT Solving has grown a lot, with one of the latest algorithms dating just a few years ago. It should not matter whether the interest is in linear real arithmetic or in non-linear real arithmetic, while having a collection this varied and with algorithms ranging from the well known *Simplex* algorithm to new procedures such like *Subtropical Satisfiability*. Satisfiability modulo theories solving is an extension of Satisfiability Checking. In the domain of Satisfiability modulo theories it is determined whether formulas of first-order logic are satisfiable or not. For this, theory solvers are introduced. Because of the importance of this domain and the very limited number of generated exercises present for SMT, it is necessary to convey to the students exercises consisting of these theory solvers, in order to understand and learn them.

This thesis aims to describe the already two mentioned algorithms, *Simplex* and *Subtropical Satisfiability*, in addition with *Interval Constraint Propagation* and to present a way to generate exercises that can be used then for teaching purposes. The quality criterias can differ from one algorithm to another and they will be presented alongside each algorithm. As an example, for the simplex procedure it is important to make sure that a minimum number of pivoting steps is present and that at the end only integer values are to be found in the tableau. Another goal of this thesis is to elucidate the difficulties that come up when generating such algorithms and future work that can be accomplished in order to improve the generating of these algorithms or to expand to a larger number of procedures.

Erklärung

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

Antoniu-Paul Filip
Aachen, den 14. Oktober 2021

Contents

Chapter 1

Introduction

With the growth of *Satisfiability Modulo Real Algebra's* library of present procedures and its presence in many domains, grows also the need of learning these procedures. In this domain *SAT* solvers are used in combination with theory solvers to solve the desirable decision problem for formulas of first-order logic. These decision problems are meant to determine if a conjunction of formulas is satisfiable or not. Theory solvers are introduced in order to deal with many kinds of formulas. These theory solvers range from being used for linear integer arithmetic formulas to non-linear real arithmetic formulas. This thesis is focusing on three theory solvers, mainly on the *Simplex* algorithm and its first stage for dealing with linear integer arithmetic procedure decisions, on *Interval Constraint Propagation* and *Subtropical Satisfiability*, with both being incomplete and approached for real arithmetic.

The Simplex algorithm is used to find a solution for the constraints, that are given as input. This would be the first step that is usually present in the Simplex approach. The second stage consists of minimizing or maximizing the solution, depending on the requirements of the problem. This part is omitted by the theory solvers in SMT, as it is not of interest. On the other hand, as mentioned before, the ICP and Subtropical Satisfiability solvers deal with non-linear real arithmetic problems. The ICP uses interval arithmetic and the extension of Newton's method to find a way to contract the intervals that contain the roots of the polynomial. Every present variable has a lower and an upper bound, and these bounds are used in the propagation step. One downside of *ICP* would be the fact that it is not complete, meaning that in some cases the algorithm might return *Unknown* as output, instead of a propagated interval. The Subtropical Satisfiability algorithm deals with multivariate polynomials and uses the *Intermediate Value Theorem* to find solutions, when having values for which the polynomials are greater and lesser then 0. It is a very fast method, but may return *Unknown* in some cases. For this reason, this two procedures can be used in combinations with other theory solvers.

The goal of this thesis is to provide these three procedures in detail and to present a way to generate exercises for them, with the question of the problem and with every step of the solutions being presented. The question of the problem is meant to explain the students what is expected of them when solving the exercises. Having generated exercises can end up being beneficial for both the students and the institute. The institute can, in matter of minutes, generate a finite but large number of exercises,

which come, as mentioned before, with the question and solution and can be transformed directly into PDF documents. Especially under the current circumstances these generated exercises can facilitate an easier way to organise online exams, which can be more than welcomed by a lot of students. Another goal would be to explain the difficulties that can occur when trying to generate such exercises and the quality requirements needed for these exercises, which can range from having a fixed number of steps before completion to having only integer values, that are easier to be calculated by students.
The contributions of this thesis are meant to help a future work in this domain of generating such kind of exercises, by explaining how they are generated and, as mentioned before, what kind of difficulties transpire. Some of these difficulties are resolved, while some are left for future work.

Starting with chapter 2 every procedure will have its own section, where the general knowledge of the algorithm will first be presented followed by the quality requirements that are expected when generating problems. The chapters will end with the generation of every one of these methods, where an example will be present, showing how these created exercises are presented in form of a PDF document. The thesis will finalize with a quick summary followed by related work present in the domain of generating exercises and future work that can be brought.

Chapter 2

Simplex

2.1 First stage

A procedure that can be used when given a set of linear real arithmetic constraints, is the simplex method. Usually, the simplex algorithm is used for finding an optimal solution to linear programming problems, but its first step is used by SMT Solvers to find a solution to the system of the linear constraints. The given input consists of a number of constraints of the form:

$$\sum_{j=1}^{n} a_{ij} x_j \ \bowtie_i \ b_j, \tag{2.1}$$

with $a_{ij}, b_j \in \mathbb{Q}$, $\bowtie_i \in \{=, \leq, \geq\}$ for $i \in \{1,...,m\}$ with m constraints. The solution of the system will constitute a convex polyhedron.

Example 2.1.1. *For the three constraints,* $x_1 + x_2 \geq 2$*;* $-2x_1 + x_2 \geq 0$ *and* $-x_1 + x_2 \geq 0$*, it can be seen in the following figure how a geometrical representation of the constraints would look like:*

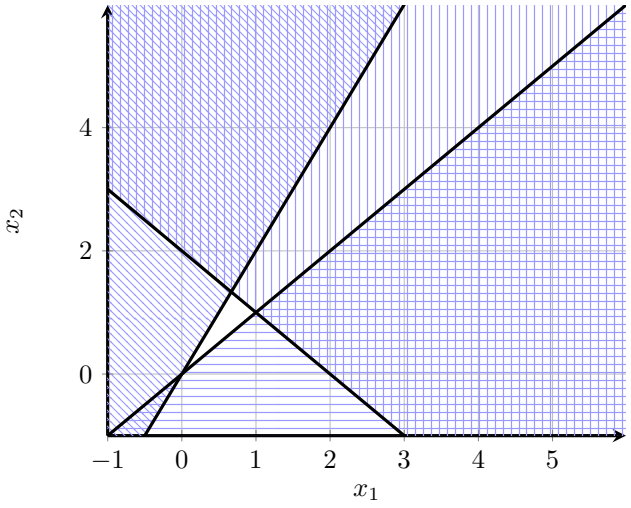


Figure 2.1: The system of constraints defining a convex polyhedron

For simplicity reasons only inequalities that are not strict are considered. Extensions of the simplex algorithm for strict inequalities have been brought, like for example in [KBD13].

First, these inequalities need to be brought to a standard form. This is done by taking the literals of the form 2.1 and adding slack variables :

$$\sum_{j=1}^{n} a_{ij}x_j - s_i = 0\ ,\ s_i \bowtie_i b_j. \tag{2.2}$$

s_1,...,s_m are the added slack variables. With the help of these slack variables the constraints will only consist of equalities. These constraints are then used for the theory solver comprising the simplex tableau. The tableau includes the two sets of *basic* B and *non-basic* N variables, with the before added slack variables being the basic variables, which build the identity matrix on the right side of the tableau. These basic variables are compiled with the help of the other free variables from the linear constraints. This is the reason why these basic slack variables are also called *dependent* variables.
Every row of the tableau consists of these equations expressing the basic variables :

$$\bigwedge_{s_i \in B} (s_i = \sum_{x_j = N} a_{ij}x_j) \tag{2.3}$$

Constraint	x_1	x_2	s_1	s_2	bound
Assignment	0	0	0	0	0
f_1	a_{11}	a_{12}	1	0	b_1
f_2	a_{21}	a_{22}	0	1	b_2

Table 2.1: Abstract form of a tableau with the dependent variables added. This form is maintained throughout every operation present.

Throughout all operations the algorithm maintains the form of the tableau and an assignment α for all variables. At first, all variables are assigned the value $\alpha(x_i) = 0$ and the basic variables are the newly added dependent variables. The dependent variables need to satisfy their bounds. The assignments mentioned before are used to determine if the variables are violating their bounds or not and will constitute the solution at the end of the procedure.
If the algorithm finds a dependent variable s_i, whose bounds are violated then it appeals to pivoting. Pivoting consists of looking for a non-basic variable x_i in the linear combination of s_i. The assignment of this found variable can be then either increased or decreased in order for the dependent variable to maintain its bounds. While pivoting, the simplex changes its form and location by using one its variables and shifting it around the hyperplane, while the values of the other variables stay untouched. After this step the variable x_i takes the role of the dependent variable s_i in the tableau, becoming a basic variable and building the identity matrix of the right side of the tableau. At the same time, x_i will be replaced in every other linear combination by s_i. After these changes, the assignment of s_i is set to the previously violated bound. The assignments of the rest of the basic variables x_j are increased

by :

$$\vartheta = \alpha(x_j) + \frac{l_i - \alpha_(x_j)}{a_{ij}}, \tag{2.4}$$

where l_i constitutes the bound and a_{ij} the coefficient present in the tableau. The algorithm repeats pivoting until no bounds are violated by the dependent variables anymore or no suitable variables for pivoting were found. If no bounds are violated with the current assignments, these assignments become the model for the formula. In case there is a violated bound present, the algorithm needs to send as output an explanation consisting of the constraints, which combined are unfeasible.
For linear inequalities consisting of integer variables the *Branch-and-Bound* method is most commonly used. At the beginning the algorithm works the same, because if there is no solution found for real variables, there won't exist any solution for the integer ones either. If there exists a solution but for at least a variable x_j the assigned value $\alpha(x_j)$ is real then the *Branch-and-Bound* algorithm will try to find a solution for an integer assignment $v \leq \lfloor \alpha(x_j) \rfloor$ or $v \geq \lceil \alpha(x_j) \rceil$.
One problem with Simplex is that cycles can occur where the same pivoting steps are executed again and again and the bases present are repeated. One easy way to avoid them is to use *Bland's Rule*, which consists of having a determined order in which variables with violate bounds are picked for pivoting, explained in [Pan90].
This concludes the first step of the Simplex algorithm. As mentioned before, the second step is two maximize or minimize depending on the requirements the solution found. Because the theory solver is only preoccupied with the existence of a solution, this step is omitted.

Constraint	x_1	x_2	bound
Assignment	0	0	0
f_1	-1	0	1
f_2	6	1	4

Example 2.1.2 (Creation of matrix)**.** *The matrix above shows an initial tableau with the bounds, initial assignments and coefficients. Every value necessary to start the algorithm is present. The first step is to bring the tableau to a standard form.*

Initially the matrix needs to be brought to the standard form. This is done by increasing the size of the matrix, in this case the number of columns by the number of rows. Every row should contain a new slack variable. These slack variables also needed to get the initial assignments and the bounds depending which row(constraint) they represent. After the preprocessing is done, the algorithm goes through these slack variables to check if any of these slack variables is violating their bounds.

Constraint	x_1	x_2	s_1	s_2	bound
Assignment	0	0	0	0	0
f_1	-1	0	1	0	1
f_2	6	1	0	1	4

Example 2.1.3 (Addition of slack variable)**.** *The tableau above shows the extended matrix after the slack variables were added. The initial assignment for the slack variables is equal to* 0*.*

After the addition of slack variables, the bounds of these variables are checked from left to right. The first slack variable that violates its bound is picked. The order can also be determined by *Bland's Rule*, one aspect that can be added in future work. Because the matrices that are used are this small, the usage of *Bland's Rule* is not a necessity. When a slack variable has been found, that violates its bound, a candidate for pivoting needs to be found. After such a candidate has been found, the bound of the variables is "fixed" and a new violating slack variable is searched for. This is repeated until no violating variables are present anymore. If this is the case, the current assignments represent a solution for the linear combination of the constraints.

Constraint	s_1	x_2	x_1	s_2	bound
Assignment	1	0	-1	-6	0
f_1	-1	0	1	0	1
f_2	-6	1	0	1	4

Example 2.1.4 (First pivot step)**.** *In this example the first violating slack variable was s_1. The candidate for the pivot step was x_1. After this step the new assignment of $\alpha(s_1) = 1$ and $\alpha(s_2) = -6$.*

Constraint	s_1	s_2	x_1	x_2	bound
Assignment	1	4	-1	10	0
f_1	-1	0	1	0	1
f_2	6	1	0	1	4

Example 2.1.5 (Second pivot step)**.** *In this example the second violating slack variable was s_2. The candidate for the pivot step was x_2. After this step the new assignments are $\alpha(s_1) = 1$ and $\alpha(s_2) = 4$. Because no more slack variable is violating its bounds, the algorithm terminates and the current assignments constitute the solution.*

2.2 Quality requirements

The most important aspect that needs to be present when dealing with the simplex algorithm is that every current coefficient can be easily read from the matrix that represents the tableau. With the help of this matrix every function present in the implementation needs to know the locations of the basis variables, the current assignments of the variables or, as mentioned before, the coefficients of the constraints. Every operation should be automated so that the functions can operate on matrices of different sizes for $n \times m$, for n,$m \in \{1,2,3\}$.

Another essential feature is a desired number of pivot steps that take place, so that the algorithm does not end, for example, after just one pivot step when the number of constraints is 3. This can be ensured that the bounds are all greater than 0, knowing that all assignments start with 0. If every bound is lesser than 0, then every constraint is trivially satisfied with the solution being 0. Remember that no strict inequalities are present. Another way to guarantee at least two steps of pivoting, is to have some coefficients that are negative. This way the updated assignments for the dependent variables don't necessarily exceed 0 even though they were not the variables present in the pivoting. In addition to this, the randomised coefficients at

the start can be picked in such a way that, for example, every dependent variable is present in a pivot step. For instance, for a matrix of the form n, the algorithm will have n steps until a feasible solution is found. This can be done when calculating 'in advance' every operation present. If this is not the case, new coefficients will be randomised. This method can, obviously, be improved, but taking in consideration how small the matrices are, the alteration will not make a big difference.

One other quality is to have the exercise automatically generated in a PDF document, with the question of the problem and solution being presented with every important step. Important aspects that can be written when presenting the solution is what candidates are chosen for the following pivot step and the succeeding tableau.

Finally, an aspect that needs to be taken into consideration when dealing with so many operations, is to ensure that throughout the procedure only integer numbers are present. Having only integer numbers simplifies the operations a lot. Again, this can be done by calculating beforehand the coefficients that will occur with the current randomised numbers. If these numbers don't satisfy this requirements, new numbers will be generated. The difficulty lies in having every or almost every operation with the current coefficients calculated beforehand. After just a few pivoting steps the number of operations can grow significantly. Here is important to ensure that no infinite loops can be created. This is guaranteed by checking on whether the desired integer property of every value is still valid. After the operations were calculated and the property is valid, the loop can end and the exercise will be presented.

2.3 Generation of exercises

It is important that the user has the ability to choose on how big the tableaus will be. This can be done by asking what the number of constraints and variables should be present.
Every exercise can start with a question that is intended to explain the students what is expected when solving the exercise. For the simplex exercises a question could look like: *"Given the following tableau, consisting of constraints, find a solution. If no solution can be found, an explanation is required as to why this is the case. "* The first step needed to generate the problem involves the matrix that will contain the coefficients of the constraints. The rows will represent the number of constrains, while the columns will symbolize the number of variables. All values can be randomised from a specific list, so that only desired numbers will be present. Besides the coefficients of the constraints, the bounds are also generated and are found on the right sides of the constraints. As mentioned before, it is important that every information needed can be read from the matrix. For this reason the first row will also contain every assignment for the variables underneath. Every tableau constitutes a step in the exercise. After every pivot step a new tableau is shown as output.
After the user decides on the number of constraints and variables the program will generate the exercise alongside the solution in a PDF document.

Exercise: Given the following tableau, consisting of constraints, bring it to the standard form and find a solution. If no solution can be found, an explanation is required as to why this is the case. The order of choosing pivot candidates can be $x_1 > x_2 > s_1 > s_2$.

Constraint	x_1	x_2	bound
Assignment	0	0	0
f_1	-1	0	1
f_2	6	1	4

Solution: Added slack variables:

Constraint	x_1	x_2	s_1	s_2	bound
Assignment	0	0	0	0	0
f_1	-1	0	1	0	1
f_2	6	1	0	1	4

The slack variable with number 1 is violating its bound, because $0 < 1$.
The candidate chosen for pivoting is the first variable x_1.

Constraint	s_1	x_2	x_1	s_2	bound
Assignment	1	0	-1	-6	0
f_1	-1	0	1	0	1
f_2	-6	1	0	1	4

The slack variable with number 2 is violating its bound, because $-6 < 4$.
The candidate chosen for pivoting is the second variable x_2.

Constraint	s_1	s_2	x_1	x_2	bound
Assignment	1	4	-1	10	0
f_1	-1	0	1	0	1
f_2	6	1	0	1	4

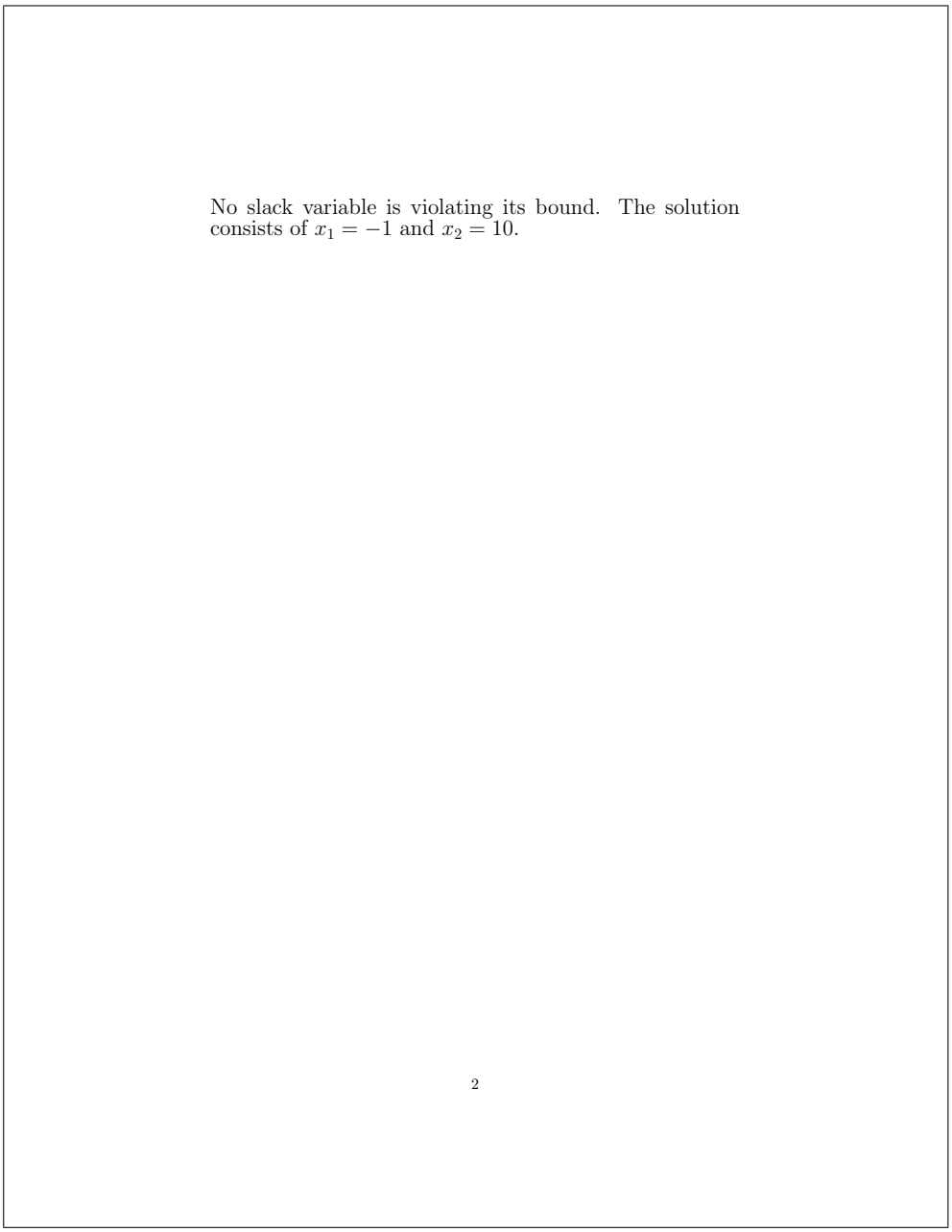


Figure 2.2: PDF document generated for a simplex exercise.

Chapter 3

Interval Constraint Propagation

Interval Constraint Propagation [VHMK97] is a procedure that uses propagation of intervals in order to solve quantifier-free nonlinear real arithmetic formulas. It is incomplete and even though the output may be 'Unknown' it can still end up being useful because of the reduction of the intervals and thus the search space. It is a procedure that can be used in combination with other theory solvers.
The theory solver Interval Constraint Propagation becomes as input a series of polynomials that can be either univariate or multivariate. Each of these polynomials are defined over interval domains. Every variable present in these polynomials has a set of bounds, specifically a *lower* and an *upper* bound. In order to deal with these intervals *Interval Arithmetic* is introduced, which is used for the addition, subtraction, multiplication and lastly, division.

3.1 Interval Arithmetic

First of all, the real arithmetic operations need to be partially extended for operation with intervals. For $+, -, \cdot, \div : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $\mathbb{R}$ will be extended to $\mathbb{R} \cup \{-\infty, +\infty\}$. For two real numbers a,$b \in \mathbb{R}$ the operations will be defined as:

Addition	$-\infty$	b	$+\infty$
$-\infty$	$-\infty$	$-\infty$	
a	$-\infty$	$a+b$	$+\infty$
$+\infty$		$+\infty$	$+\infty$

Table 3.1: Table consisting of addition arithmetic with $-\infty$ and $+\infty$

Subtraction	$-\infty$	b	$+\infty$
$-\infty$		$-\infty$	$-\infty$
a	$+\infty$	$a-b$	$-\infty$
$+\infty$	$+\infty$	$+\infty$	

Table 3.2: Table consisting of subtraction arithmetic with $-\infty$ and $+\infty$

Multiplication	$-\infty$	$-\infty < b < 0$	0	$0 < b < +\infty$	$+\infty$
$-\infty$	$+\infty$	$+\infty$	0	$-\infty$	$-\infty$
$-\infty < a < 0$	$+\infty$	$a \cdot b$	0	$a \cdot b$	$-\infty$
0	0	0	0	0	0
$0 < a < +\infty$	$-\infty$	$a \cdot b$	0	$a \cdot b$	$+\infty$
$+\infty$	$-\infty$	$-\infty$	0	$+\infty$	$+\infty$

Table 3.3: Table consisting of multiplication arithmetic with $-\infty$ and $+\infty$

Division	$-\infty$	$-\infty < b < 0$	0	$0 < b < +\infty$	$+\infty$
a	0	$a \div b$		$a \div b$	0

Table 3.4: Table consisting of division arithmetic with $-\infty$ and $+\infty$

It is to be observed that the tables above are only partial. Cases that don't show up in the tables, don't appear in any exercises either and are not needed.

An interval A is a closed and connected subset of $\mathbb{R}$, which can be defined as follows:

$$A = [\underline{A}\,;\,\overline{A}] = \{v \in \mathbb{R} \mid \underline{A} \leq v \leq \overline{A}\}. \tag{3.1}$$

$\underline{A} \cup \mathbb{R} \cup \{-\infty\}$ denotes the *lower bound* and $\overline{A} \cup \mathbb{R} \cup \{+\infty\}$ the *upper bound*. This interval A is called *bounded* if both the lower and upper bounds are unequal $-\infty$ and $+\infty$. Is this not the case, they are called *unbounded*. In case of the presence of point intervals, [v,v] is defined as $\{v\}$ and can be written simply as v, where $v \in \mathbb{R}$.

The operations needed will be presented as in [Kul09]. As mentioned before, when dealing with intervals, real arithmetic needs to be extended to *Interval Arithmetic*. The semantics of every used operator is defined below:

$$+, -, \cdot, \div : \mathbb{I} \times \mathbb{I} \to \mathbb{I}, \tag{3.2}$$

where $\mathbb{I}$ is the set of all intervals.

Definition 3.1.1 (Interval Addition)**.** *The addition of two intervals A, B with $A = [\underline{A}\,;\,\overline{A}]$ and $B = [\underline{B}\,;\,\overline{B}]$ is defined as $A + B = [\underline{A} + \underline{B}\,;\,\overline{A} + \overline{B}]$ or $A + B = \emptyset$ in case either A or B is empty.*

Definition 3.1.2 (Interval Substraction)**.** *The addition of two intervals A, B with $A = [\underline{A}\,;\,\overline{A}]$ and $B = [\underline{B}\,;\,\overline{B}]$ is defined as $A - B = [\underline{A} - \overline{B}\,;\,\overline{A} - \underline{B}]$ or $A - B = \emptyset$ in case either A or B is empty.*

Definition 3.1.3 (Unary interval minus)**.** *$-A$ is defined as $0 - A = [-\overline{A}\,;\,-\underline{A}]$ for non-empty Interval $A = [\underline{A}\,;\,\overline{A}]$.*

Definition 3.1.4 (Interval Multiplication)**.** *The multiplication of two non-empty intervals A, B with $A = [\underline{A}\,;\,\overline{A}]$ and $B = [\underline{B}\,;\,\overline{B}]$ is defined as $A \cdot B = [min(\underline{A} \cdot \underline{B}\,,\,\underline{A} \cdot \overline{B}\,,\,\overline{A} \cdot \underline{B}\,,\,\overline{A} \cdot \overline{B})\,;\,max(\underline{A} \cdot \underline{B}\,,\,\underline{A} \cdot \overline{B}\,,\,\overline{A} \cdot \underline{B}\,,\,\overline{A} \cdot \overline{B})]$ or $A \cdot B = \emptyset$ in case either A or B is empty.*

Example 3.1.1. *For two intervals $A = [-2; 3]$ and $B = [3; 5]$ their multiplication is $A \cdot B = [-6; 15]$.*

Because this procedure deals with polynomials that are not necessary linear, the case where an interval is multiplied by itself needs to be taken into consideration.

Definition 3.1.5 (Interval Square)**.** *For a non-empty interval $A = [\underline{A}\,;\,\overline{A}]$ the square of A is defined as $A^2 = (A \cdot A) \cap [0\,;\,+\infty]$.*

Example 3.1.2. *For an interval $A = [-2; 2]$ its square is equal to $[-2; 2]^2 = [0; 4]$.*

Definition 3.1.6 (Interval Division)**.** *The division of two intervals non-empty A, B with $A = [\underline{A}\,;\,\overline{A}]$ and $B = [\underline{B}\,;\,\overline{B}]$ is defined as $A \cdot \frac{1}{B} = A \cdot [\frac{1}{\overline{B}}\,;\,\frac{1}{\underline{B}}]$, where $0 \notin B$.*

In case $0 \in B$, the possible results can be depicted from table 3.5. Note that the interval that results can contain a gap.

$A \div B$	$B = [0; 0]$	$\underline{B} < \overline{B}$	$\underline{B} < 0 < \overline{B}$	$0 = \underline{B} < \overline{B}$
$0 \in A$	$(-\infty\,;\,+\infty)$	$(-\infty\,;\,+\infty)$	$(-\infty\,;\,+\infty)$	$(-\infty\,;\,+\infty)$
$\overline{A} < 0$	$\emptyset$	$[\overline{A}/\underline{B}\,;\,+\infty]$	$[-\infty; \overline{A}/\overline{B}] \cup [\overline{A}/\underline{B}\,;\,+\infty]$	$[-\infty\,;\,\overline{A}/\overline{B}]$
$0 < \underline{A}$	$\emptyset$	$[-\infty\,;\,\underline{A}/\underline{B}]$	$[-\infty; \underline{A}/\underline{B}] \cup [\underline{A}/\overline{B}\,;\,+\infty]$	$[\underline{A}/\overline{B}\,;\,+\infty]$

Table 3.5: Table consisting of possible cases of dividing when $0 \in B$

3.2 ICP Algorithm

The aim of the procedure is to find the roots of a polynomial. The interval that contains the roots proceeds to be propagated, such that the search space becomes smaller. A method to do such a thing is the *Extension of Newton's Method*. The Interval Constraint Propagation Module becomes as input a conjunction of inequations of form $p \bowtie 0$ with $\bowtie \in \{<\ ,\ \leq\ ,\ \geq\ ,\ >\}$ and where p is an univariate or multivariate polynomial, which is continuously differentiable. On these inequations some preprocessing steps are required such that the linear and non-linear polynomials are separated from each other and that the inequations are transformed in equations. In this case of generation of exercises only univariate polynomials are taken into consideration and only one step of a contraction is performed on each polynomial. Because of this, the preprocessing is omitted. More information on how the precprocessing is done and how the algorithm works for multivariate polynomials is to be found in [SÁRL13]. The algorithm itself consists of choosing randomly or by certain heuristics a *contraction canditate*, on which the contraction is performed. A *contraction canditate* can be comprised of a polynomial p and a variable x present in p. The contraction step is made with the help of *Newton's extended method*.

3.2.1 Extension of Newton's method

Newton's method is a root finding algorithm, which consists of producing successively approximations of the roots of a polynomial. The approximations need to get closer to the root with every step. If this is not the case, then the method is diverging, something that can happen often when dealing with linear polynomials, and the

approximation needs to be stopped. The method starts with a chosen starting point s_0 and the values of the polynomial and its derivative in this point s_0. In case the polynomial is multivariate and consists, for example, of n variables, then by assigning values to $n-1$ variables, the resulting polynomial becomes univariate and the method can be applied. By having the starting point s_0

$$s_1 := s_0 + \frac{f(s_0)}{f'(s_0)} \tag{3.3}$$

will be a better approximation of the root than s_0 was. This step is continued until an appeasable approximation is found. Note that $f'(s_0)$ denotes the derivative of the polynomial f in point s_0.

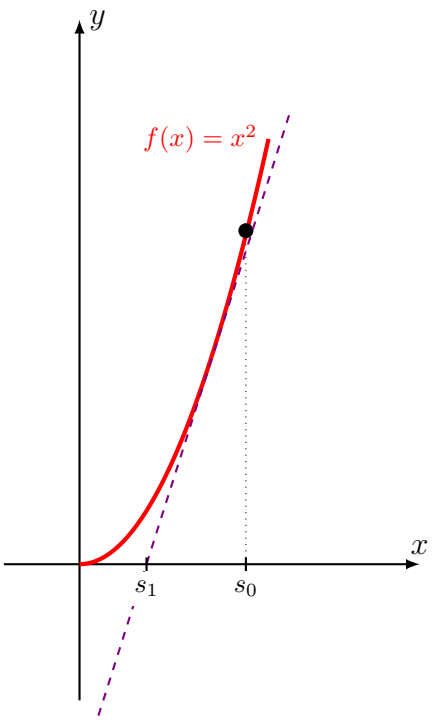


Figure 3.1: A function $f(x) = x^2$ and its derivative $f'(x)$ in a starting point s_0. The new point s_1 will be a better approximation for the root.

This method is extended in order to cope with interval-valued polynomials. It was presented by [HR97]. This extension makes use of the first-order version of Taylor's theorem, which denotes that

$$\forall s,x \in A \, \exists \xi \in A : f(x) = f(s) + (x + s) \cdot f'(\xi), \tag{3.4}$$

where f is a interval-valued polynomial, x a variable present in f, A a starting interval for x and s can denote the center point in interval A. In case f is a multivariate

polynomial A becomes a set box of intervals for every variable in f, but, as stated before, in this case only univariate polynomials are taken into consideration when generating them. Assuming that $f'(\xi)$ is not equal to 0, transforming the equation from above for a root x of f, then:

$$x = s - \frac{f(s)}{f'(\xi)}. \tag{3.5}$$

By replacing ξ with the whole interval A, it results the following equation:

$$A' = s - \frac{f(s)}{f'(A)}. \tag{3.6}$$

Note that if $f'(A)$ is not equal to zero, but carries a zero, the resulting interval A' will contain a gap and it will be calculated as in Table 3.5.

Example 3.2.1. *Let $f = x^2 - 2x + 1$ with its root in 1 with a starting interval $A = [0,3]$ and a sample point $s = 3$. Having the polynomial and A, the evaluation of the set of derivatives of f in A can start, where $f' = 2x - 2$.*

$$f'(A) = [2;2] \cdot [0;3] - [2;2] = [-2;4].$$

The next step is to evaluate the polynomial f for the sample point s_0 :

$$f(3) = 3^2 - 2 \cdot 3 + 1 = 4.$$

After this, the contraction step can begin :
The set of possible zeros in A :

$$s - \frac{f(s)}{f'(A)} = [3,3] - \frac{[4;4]}{[-2;4]} = [-\infty;2] \cup [4;+\infty].$$

Then new contracted interval A' will be:

$$A' = [0;3] \cap ([-\infty;2] \cup [4;+\infty]) = [0;2].$$

3.3 Quality requirements

As mentioned before, because of simplicity reasons, only univariate polynomials are taken into consideration. In addition to this only one step of propagation is done on every polynomial present.
The most important aspect when generating such polynomial root finding problems is to know beforehand these roots. In a similar way on how the coefficients are generated for the simplex algorithm, the roots of every polynomial will be stored in a matrix, where every row represents a polynomial. With the help of *Vieta's formulas* the coefficients of these polynomials can be calculated. By knowing the interval on which the roots are found, the starting interval for Newton's method A can be produced, for example by containing all but one root. If the polynomial possesses only one root, then that root should be included. Another benefit of knowing the roots in advance is to evaluate how good the propagation at the end is.

Vieta's formulas can be used to construct the coefficients of a polynomial with the help of the roots. By the *fundamental theorem of algebra* every polynomial of grad n is set to have n complex roots. These roots don't have to be distinct.

Definition 3.3.1 (Vieta's Formulas)**.** *For a polynomial* $f = a_x x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$*, with roots* $r_1, r_2, ..., r_n$*, the Vieta's Formulas can be written in a condensed way as :*

$$\sum_{1 \le i_1 \le i_1 \le ... \le i_k \le n} (r_{i_1} \cdot r_{i_2} \cdot ... \cdot r_{i_k}) = (-1)^k \frac{a_n - k}{a_n}.$$

The indeces ensure that the roots are taken i at the time.

Keeping an eye on the time and space complexity is also very important. As it will be mentioned in the future work chapter of this thesis, the implementations of every algorithm is far from perfect. Because *ICP* contains very many operations it is important that the generating does not take too long. For this reason some methods can be implemented together. For example, evaluating an interval in a function and constructing the derivative of this function can be done in the same loop of the same method. As it can be observed in Example 3.2.1 the only time the evaluating of a function for an interval is necessitated is for the the set of derivatives of the polynomial for the given interval.

The precision of the operations is also very important. Because of this, the *GMP Library* was used, which is a precision arithmetic library. This library does not have a predefined value for $infinity$, so one had to implemented. By doing this, also the operations need to be overloaded in order to cope with the newly defined $infinity$ value. The most important operators are overloaded the same way the real arithmetic is expanded for $infinity$.

The biggest difficulty present when dealing with interval arithmetic comes from the fact that for two intervals $A = [\underline{A}; \overline{A}]$ and $B = [\underline{B}; \overline{B}]$ the result of $[\underline{A}; \overline{A}] + [-\underline{B}; -\overline{B}]$ differs from $[\underline{A}\,; \overline{A}] - [\underline{B}\,; \overline{B}]$. Because of the way the coefficients are stored in a matrix, it is important for every method to know how to deal with a negative value. For example, if a coefficient is $a < 0$, then the methods that deal with interval arithmetic will interpret it as a point interval $-[a; a]$, and not as $+[-a; -a]$ or $-[-a; -a]$.

As by the simplex algorithm, a question comes with every generated exercise in order to explain what is meant to be calculated. Every exercise comes with the solution, consisting of the most important steps when solving the problem. Again, as by the Simplex procedure, the exercises are generated as PDF Document.

3.4 Generation of exercises

Every operation needs to be overloaded in order to work with intervals. Intervals can be constructed with the help of a structure, which contains a *lower* and an *upper* bound. The generation starts with the number of polynomials needed, plus the number of roots for every polynomial. As mentioned before, every root is stored as an coefficient on a row in a matrix.

Example 3.4.1. *If the user intends to have two polynomials, both with* 3 *roots, then the following matrix will be generated :*

Polynomial	r_1	r_2	r_3
f_1	1	2	0
f_2	0	1	-2

Having the roots, the following polynomials will be constructed :

$$f_1 = x^3 - 3x^2 + 2x, \text{ and } f_2 = x^3 + x^2 - 2x.$$

Because every value is read from finite lists containing the possible values for the roots, it can happen that a polynomial contains a root more than once. In that case, even though the desired number of roots was d, the polynomial will contain only $d-1$ roots. With the help of the values in this matrix and *Vieta's Formulas* the coefficients of every polynomial is calculated and stored in a new matrix. Again, every row will constitute a polynomial and every column will represent the power.

Example 3.4.2. *For the example from before, the following matrix would be created, which constitutes the two polynomials :*

Polynomial	x^0	x^1	x^3	x^4
f_1	0	2	-3	1
f_2	0	-2	1	1

After having the polynomials constructed and the roots, a sample point s_0 and a starting interval A are chosen in accordance with the roots. Having every important information needed, the algorithm can start with calculating the derivative of the polynomial for the interval A. The other value needed is the function value for point s_0. Having every value needed the generated exercise in form a PDF Document can be constructed :

Exercise: Given the following polynomial $f = x^2 - 2x + 1$, with the constraint $f = 0$, with the starting interval $A = [0, 3]$ and a sample point $s = 3$, perform one step of constraint propagation using the *Extension of Newton's method*:

Solution:

$$f' = 2x - 2.$$

The set of derivatives of f is :

$$f'(A) = [2; 2] \cdot [0; 3] - [2; 2] = [-2; 4].$$

Evaluation of f in point $s_0 = 3$:

$$f(3) = 3^2 - 2 \cdot 3 + 1 = 4.$$

The set of possible zeros in A :

$$s - \frac{f(s)}{f'(A)} = [3, 3] - \frac{[4; 4]}{[-2; 4]} = [-\infty; 2] \cup [4; +\infty].$$

Then new contracted interval A' will be:

$$A' = [0; 3] \cap ([-\infty; 2] \cup [4; +\infty]) = [0; 2].$$

1

Chapter 4

Subtropical Satisfiability

4.1 Subtropical Satisfiability algorithm

Subtropical Satisfiability is a fast method that finds a strictly positive solution to constraints of the form

$$f \bowtie_i 0, \textit{with } \bowtie_i \in \{>, \geq, =\}, \tag{4.1}$$

where $f \in \mathbb{Z}[x_1,...,x_n]$ with $x = (x_1,...,x_n)$ as a vector of variables, is multivariate polynomial. The moment a solution is found, the procedure returns it, but in case no solution was found, the constraints may still hold. Because of this Subtropical Satisfiability is incomplete. For this reason, as by ICP, it can be used in combination with other theory solvers. As a procedure, it was first explained in [Stu15] and then improved a few years later in [FOSV17].

The method starts with evaluating the polynomial in point 1, with the hope that $f(1,...,1) < 0$. If $f(1,...,1) = 0$ and $f = 0$ was given as a constraint, Subtropical Satisfiability can return 1 as a solution. If $f(1,...,1) > 0$ the method considers $-f$ instead. With $f(1,...,1) < 0$, the procedure tries to find a positive solution p, such that $f(p) > 0$. It uses then the *Intermediate Value Theorem*, which states, that a continues function, with a positive and negative value, contains a *zero*.

Definition 4.1.1 (Frame of a polynomial)**.** *For $f \in \mathbb{Z}[x_1,...,x_n]$ with $x = (x_1,...,x_n)$ it stands:*

$$frame(p) = \{p_i \mid i \in \{1,...,n\}\} \textit{ for } \textstyle\sum_{i=1,2,...n} f_{p_i} x_i^{p_i}.$$

Every frame p_i can be partitioned into positive and negative frames, according to the sign that f_{p_i} has.

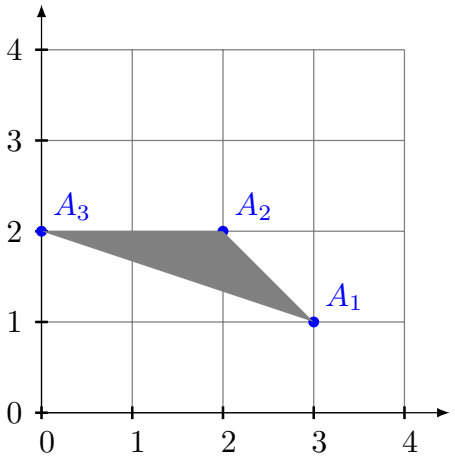
$$frame^+ = \{p_i \in frame(p) \mid f_{p_i} > 0\}, frame^- = \{p_i \in frame(p) \mid f_{p_i} < 0\}. \tag{4.2}$$

Example 4.1.1. $f = 3x^3y - 4y^2 + 2x^2y^2$, *with* $frame(f) = \{(3,1);(0,2);(2,2)\}$. *The positive and negative frames are then chosen based on the sign of the coefficient :* $frame^+(f) = \{(3,1);(2,2)\}$ *and* $frame^-(f) = \{(0,2)\}$.

Definition 4.1.2 (Convex hull). *For $S \in \mathbb{R}^d$ $conf(S) \subseteq \mathbb{R}^d$ is the unique inclusion-minimal convex that contains S.*

With the help of the convex hull, the Newton polytope can be defined:

Definition 4.1.3 (Newton polytop). *The Newton polytope of a polynomial f is the convex hull of its frame, such that $newton(f) = conv(frame(f))$.*



Example 4.1.2. *For $f = 3x^3y - 4y^2 + 2x^2y^2$ the above graph shows the newton polytope of its frames.*

Definition 4.1.4 (Face of polytop). *Let $P \subseteq \mathbb{R}^d$ be a polytop. A face of P with respect to a vector $n \in \mathbb{R}^d$ is*

$$face(n,P) = \{p \in P \mid n^T p \geq n^T q\} \text{ for every } q \in P.$$

Every face that has a dimension of 0 is called a *vertex*. $V(P)$ denotes all vertices in P. $p \in V(P)$ if and only if $n^T p > n^T q$ for all $q \in P \setminus \{p\}$, for a vector $n \in \mathbb{R}^d$. Because $frame(f)$ is always finite, it holds that

$$V(frame(f)) \subseteq frame(f) \subseteq newton(f).$$

Lemma 4.1.1. *Be $S \subset \mathbb{R}^d$ finite and $p \in S$. Then it holds that the two following statements are equivalent:*
(1.) p is a vertex of $conv(S)$ for a vector $n \in \mathbb{R}^d$.
(2). A hyperplane H can be found, such that $n^T x + c = 0$ that strictly separates from any other $q \in S$. The normal vector n is directed from H towards p.

Having found such a $face$ $p \in conv(S)$ and a hyperplane H that separates it from the $q \in conv(S) \setminus \{p\}$, this $face$ will dominate the entire polynomial in the direction of the normal vector n with $n \in \mathbb{R}^d$. With the help of a such hyperplane p will be found, with all positive coordinates, such that $f(p) > 0$.

Lemma 4.1.2. *Let f be a multivariate polynomial and $p \in frame(f)$ a vertex of $newton(f)$ with respect to a vector $n \in \mathbb{R}^d$. Then for all $a \in \mathbb{R}^+$ there is a $a_0 \in \mathbb{R}^+$, with $a \geq a_0$, such that :*

$$(1.)\ |f_p a^{n^T p}| > |\textstyle\sum_{q \in P \setminus \{p\}} f_q a^{n^T q}| \text{ and}$$

$$(2.)\ sign(f(a^n)) = sign(f_p).$$

All that is needed to find a point p with all positive coordinates, such that $f(p) > 0$, is to find a $p \in frames^+(f)$, which is a vertex of $frames(f)$ with respect to a vector $n \in \mathbb{R}^d$. If such a point p is found the second part of the lemma assures that $sing(f(a^n)) = sing(f_p) = 1$, for a $a \in R^+$ that is large enough. If $sing(f(a^n)) = sing(f_p) = 1$ holds then $f(a^n) > 0$ and thus, the point with all positive coordinates has been found. The problem can be brought down to having such a $p \in frames^+(f)$ and finding a hyperplane $H : n^T x + c = 0$ that separates it from the $frame(f) \setminus \{p\}$. Let $frame(f) \subset \mathbb{R}^d$ and $n \in \mathbb{R}^d$, then the problem can be encoded as:

$$\varphi(p,frame(f),n,c) = (n^T p + c > 0 \ \wedge \bigwedge_{q \in frames(f) \setminus \{p\}} n^T q + c < 0), \tag{4.3}$$

where $p \in frame^+(f)$. This is a *linear* problem with $d + 1$ real variables n and c. This problem can be solved with an extension of the *simplex* algorithm in such a way that it can work with strict inequalities. Because the interes is in finding the entire solution set, the *linear* problem will be solved, in this case, with the help of *Fourier-Motzkin* method.

After having $f(1,...,1) < 0 < f(p)$, where p consists of real coordinates, the root can be found on the straight line from (1,...,1) to p. This consits of three steps. First, a new univariate polynomial f' is to be constructed from the original multivariate polynomial f, by parameterising all variables in a new variable $t \in [0; 1]$, such that the straight line mentioned above is crossed. Then a root t_0 is to be found for f' with the help of bisection, for example. Finally, the root of f can be constructed as a point on the straight line from (1,...,1) to p for the parameter t_0 found in the second step.

4.2 Fourier-Motzkin

Fourier-Motzkin is the earliest method for solving linear inequality systems. It was discovered by Fourier in 1826 and then re-discovered by Motzkin in 1936. It evolves around the idea of variable elimination. After a variable is picked and eliminated, an equisatisfiable formula materializes that does not contain the picked and eliminated variable anymore. This step is repeated until all variables have been eliminated. The method makes use of the *lower* and *upper* bounds of the variable to create inequalities that do not contain that variable. For a variable x_n the constraints can be subdivided in constraints that put no bound on x_n, in constraints that constitute a lower bound for x_n and constraints that compose an upper bound.
For an inequality of the form $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i$ and a variable x_n it stands that: $a_{in} \cdot x_n = b_i \cdot \sum_{j=1}^{n-1} a_{ij} \cdot x_j$. According to the coefficient a_{in} the the constraints can, as mentioned before, constitute a lower or an upper bound for x_n:

$$\begin{aligned}
&(1) : a_{in} > 0 \Rightarrow x_n \leq \frac{b_i}{a_{in}} - \sum_{j=1}^{n-1} \frac{a_{ij}}{a_{in}} \cdot x_j \ (\textit{upper bound}),\\
&(2) : a_{in} < 0 \Rightarrow x_n \geq \frac{b_i}{a_{in}} - \sum_{j=1}^{n-1} \frac{a_{ij}}{a_{in}} \cdot x_j \ (\textit{lower bound}).
\end{aligned} \tag{4.4}$$

Let β_l and β_u be the lower bound and, respectively, the upper bound for variable x_n. For each pair of this kind, a new constraint can be added in the system such that $\beta_l \leq \beta_u$. As explained before, this is done for every variable until the last constraint comprises the solution of the system. This method works also if the linear problem has both *strict* and *non-strict* inequalities, but because the *Fourier-Motzkin* algorithm will only be used for the *Subtropical Satisfiability* method, which contains only *strict* inequalities, that part will be omitted.

4.3 Quality requirements

The user can choose on how long the polynomial created will be by giving as input the number of terms that the polynomial will have. By doing this, the difficulty of the problem can vary, because the linear problem, with which the set of hyperplanes is found, becomes bigger. Another aspect, that can influence the difficulty of an exercise for *Subtropical Satisfiability*, is the form of the constraint. If, for example, the constraint created is of the form $f > 0$, the problem is solved by just finding a point for which the polynomial in that point is bigger than *zero*.

The generated exercises will only contain one polynomial with two variables of the form f_{xy}. Because of this, every information needed for this kind of a multivariate polynomial can be stored, again, in a matrix. The generated numbers can be randomised from different lists, depending on their role, for example, if they are to represent the power of the variables or just coefficients. The reason behind it is, that the powers should, in this case, not be negative and instead of randomising the numbers until every value is positive, it would be much easier and convenient to just have a second list, which is only used for the powers. The only input given is the number of coefficients desired. Having the number of coefficients the matrix can be constructed. The first value from the first column will be given from a list consisting of three numbers $-1, 0$ and 1. This values will represent the relation of the polynomial with 0. If the value is, for example, -1 it means that the constraint is of the form $f > 0$. 0 will represent the equality and 1 will be used for $\geq$. The rest of the column will remains *zero*. The first row will represent the coefficients, the second one will represent the powers of the first variable x and the third the powers for the second variable y.

Coefficient	-1	-3	2
x	0	3	0
y	0	4	3

Example 4.3.1. *The generated matrix above is representing the constraint* $-3x^3y^4 + 2y^3 > 0$. *This matrix contains every information needed in order to start the generated exercise.*

After the matrix is created it is checked, depending on whether $f(1,1) < 0$, if the polynomial needs to be inverted. The inversion consists of only switching the negative frames with the positive ones.
The second step consists of saving a current positive frame in an array, such that a new matrix can be constructed as in 4.3. With the help of this new matrix the linear problem will be solved.

Example 4.3.2. *For a constraint of the form* $0 = -3x^2y^4 + 4x^3y^3 + 2x^1y^2$ *the following linear problem would be created. Note that,* $f(1,1) = 3$*, for* $f = -3x^2y^4 + 4x^3y^3 + 2x^1y^2$*. Because of this, the polynomial* $-f$ *is taken into consideration :*

bound		px	py	c
0	$<$	2	4	1
0	$>$	3	3	1
0	$>$	1	2	1

In this example the linear problem tries to find the set of hyperplanes that separate the frame (2,4)*.*

The matrix will then be brought to a triangular form. In this way, at the end, the solution set can be read directly from the matrix. In order to create a hyperplane, values should be picked for c, followed by p_y and then p_x. The values need to respect the constraints. The correctness is guaranteed by building the correct linear problem with the help of a matrix and then bringing the matrix into the correct triangular form.

Again, as by the *Simplex* and *ICP* procedures, the exercises should be created in a PDF Document. This PDF Document will come with the question, that explains the students what is expected to be calculated and with the solution, which will consist of every important operation that takes place in this procedure.

4.4 Generation of exercises

As mentioned before, the generation starts with the number of terms in the polynomial desired. With this number, the matrix containing every important value will be generated. The coefficients can be randomised from different lists, depending on the role of the coefficient. After evaluating the polynomial in point 1, the positive and negative frames are created. The program will proceed in trying to find a solution set for the existence of a hyperplane separating a positive frame from the rest of the frames. The positive frame with the biggest coefficients will be chosen. By doing this, the probability that this frame can be separated is bigger. Note that, for example in figure 4.1.2, if a frame were to exist in the middle of the polytope, then that frame would not be separable.
After the linear problem is solved, the PDF Document including the exercise can be created.

Exercise: For the following polynomial f over the real domain and constraint, find the solution set for a hyperplane separating a positive frame p from the rest of the $frame(f)$:

$$0 = -3x^2y^4 + 4x^3y^3 + 2x^1y^2$$

Solution: $f(1,1) = 3$. Because the value of f in point 1 is greater than zero, $-f$ will be considered.
The set of positive frames is

$$frames^+(f) = \{(2,4)\}$$

The set of negative frames is

$$frames^-(f) = \{(3,3),(1,2)\}$$

The positive frame, for which a separating hyperplane is searched is $(2,4)$. The linear program for finding a hyperplane is constructed :

bound		px	py	c
0	$<$	2	4	1
0	$>$	3	3	1
0	$>$	1	2	1

Bringing the matrix in a triangular form :

bound		px	py	c
0	$<$	2	4	1
0	$>$	0	-3	-0.50
0	$>$	0	0	0.50

$$0.50 \cdot c < 0 \Rightarrow c < 0 \tag{1}$$

$$-3 \cdot py - 0.50 \cdot c < 0 \Rightarrow py > -0.16c \tag{2}$$

$$2 \cdot px + 4 \cdot py + c > 0 \Rightarrow px > -2 \cdot py - 0.50 \cdot c \tag{3}$$

The three inequalities constitute the solution set. Picking a value for c, followed by py and then followed by px, will represent a hyperplane separating the positive frame for the rest.

Chapter 5

Conclusion

5.1 Summary

The aim of this thesis was to present a way of generating exercises for algorithms often used in the domain of *Satisfiability modulo theories* for students. These exercises are meant to help the students understand the fundamentals of every algorithm by consisting of problems that only contain easy constraints. These exercises include almost every operation that can be present in the three algorithms. As explained before, every example comes with the question, that explains the student what is expected to be calculated and, most important, with the solution. Although the number of exercises that can be generated in this manner is limited, the size of the list that contains the integer values, with which the exercises are generated, guarantees that a fair amount of them can be created. Because of this amount, these exercises can be easily generated and used, for example, for online exams, that are very common because of the current circumstances and preferred by many students. Because some aspects of this generation are not perfect and because not every algorithm used in *SMT* is present in this thesis, the opportunity for future work arises. This can consist of improving the complexity on how the exercises are generated, of generating more algorithms or of rising the number of possible exercises created.

5.2 Related work

As mentioned before, there are not a lot of works that handle generation of exercises, especially for the domain of *SAT* and *SMT* solving. Prof. Kovács from the Vienna University of Technology (TU Wien) has been publishing a few papers with the goal of presenting ways to generate exercises for *SAT/SMT Solving* problems. This year a paper was published, [HKR21], which introduced a way of generating this kind of exercises meant for online exam sheets. In this paper it is explained how for generating these exercises, values are stored and randomised with the help of lists, the same way on how the exercises of the procedures, presented in this thesis, are generated.
There exists, on the internet, a number of online tools meant to solve linear problems. Some of this tools can also randomise the starting values and thus, generating a linear problem. A great online tool for solving and generating Simplex problems is the *Atozmath.com*, made by Piyush N Shah. This site can also generate polynomials for

which the roots are searched, with the help of the Newton's method, for example.

5.3 Future work

As mentioned before, the generated algorithms can be improved, especially when it comes to the complexity. For the simplex algorithm, when randomising the starting coefficients, the process starts over when at some point in the future, with the given values, a number is not integer. A more efficient way to implement this would be to store the coefficients that are randomised throughout the algorithm and only to change the values, for which the integer property is violated. For example, if at the end a value is not integer because of the last coefficient randomised, then the algorithm should not start from the beginning randomising the entire tableau, but only this last coefficient. The problem is, that the tableaus are from the start very small and the time complexity would not change a lot. A second problem would be the difficulty in implementing such thing. The program should be able to tell that only because of this one coefficient, the integer property is violated. More obvious improvements can be brought to the other two algorithms, such as *ICP* or *Subtropical Satisfiability*. In these two algorithms, at least two matrices are created to store information needed. For example, for the *ICP* method, the program starts with a matrix that contains only the the roots of the polynomials and then proceeds in creating a new matrix for the coefficients. On top of this, throughout the implementations there are a lot of temporal variables created to store values. So from the aspect of space complexity the algorithms can also be improved.
Another obvious aspect, that can be improved, is the addition of more procedures currently presented in the domain of *SMT*. These algorithms can vary from methods that are used in combination with other theory solvers such as *Branch and Bound* or *Fourier-Motzkin* to complicated ones such as *Virtual Substitution* [Wei88] or *CAD* [Col75].
Finally, another improvement, that was brought up in the summary of the thesis, is the raise of number of exercises that can be generated. Currently the algorithms use, for generating these problems, lists of values, lists that range from 5 to 10 values. Increasing the number of these values or implementing a better way of randomising them, can increase the number of exercises that can be generated.

Bibliography

[Col75] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata theory and formal languages*, pages 134–183. Springer, 1975.

[FOSV17] Pascal Fontaine, Mizuhito Ogawa, Thomas Sturm, and Xuan Tung Vu. Subtropical satisfiability. In *International Symposium on Frontiers of Combining Systems*, pages 189–206. Springer, 2017.

[HKR21] Petra Hozzová, Laura Kovács, and Jakob Rath. Automated generation of exam sheets for automated deduction. In *International Conference on Intelligent Computer Mathematics*, pages 185–196. Springer, 2021.

[HR97] Stefan Herbort and Dietmar Ratz. *Improving the efficiency of a nonlinear-system-solver using a componentwise newton method.* Inst. für Angewandte Mathematik, 1997.

[KBD13] Tim King, Clark Barrett, and Bruno Dutertre. Simplex with sum of infeasibilities for smt. In *2013 Formal Methods in Computer-Aided Design*, pages 189–196. IEEE, 2013.

[Kul09] Ulrich W Kulisch. Complete interval arithmetic and its implementation on the computer. In *Numerical Validation in Current Hardware Architectures*, pages 7–26. Springer, 2009.

[Pan90] Ping-Qi Pan. Practical finite pivoting rules for the simplex method. *Operations-Research-Spektrum*, 12(4):219–225, 1990.

[SÁRL13] Stefan Schupp, Erika Ábrahám, Peter Rossmanith, and Dipl-Inform Ulrich Loup. Interval Constraint Propagation in SMT compliant decision procedures. *Master's thesis, RWTH Aachen*, 2013.

[Stu15] Thomas Sturm. Subtropical real root finding. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 347–354, 2015.

[VHMK97] Pascal Van Hentenryck, David McAllester, and Deepak Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of symbolic computation*, 5(1-2):3–27, 1988.